Digital Equipment Corporation
Maynard, Massachusetts

digital

PDP-12
User's Manual

FPP-12

Floating Point Processor

DEC-12-GQZA-D

FPP-12
FLOATING POINT PROCESSOR USER'S MANUAL

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

1st Edition December 1970
2nd Edition April 1971

CONTENTS

CONTENTS (Cont)

ILLUSTRATIONS

ILLUSTRATIONS (Cont)

TABLES

CHAPTER 1
FPP12 PROGRAMMING DATA

1.1 SCOPE OF THE MANUAL

This manual contains instructions for programming and servicing the Floating Point Processor (FPP12), a peripheral device for PDP-8 and PDP-12 Computers. Chapters 1 and 2 contain programming information. Chapter 3 describes the logical components of the FPP12. Installation and check-out procedures are found in Chapter 4. It is assumed that the reader is familiar with the operation of the PDP-8/I, PDP-8/L, PDP-8, LINC-8, PDP-8/E or PDP-12 Computers. It is suggested that the reader have as a reference the FPP Assembler Manual.

1.2 INTRODUCTION

The Floating Point Processor (FPP12) is a programmable, peripheral, digital processor that attaches to the input/output (I/O) bus of any PDP-8, PDP-8/I, PDP-8/L, PDP-8/E, LINC-8, or PDP-12 Computer. The FPP12 is a parallel processor with its own instruction set. It utilizes the direct memory access or data break facility to "steal" memory cycles from the Central Processor Unit (CPU). Similar to a disk, the FPP12 is activated by the CPU through the use of programmed input/output IOT instructions. Once activated, the FPP12 steals a maximum of 50 percent of the memory cycles from the PDP-8, PDP-8/I, PDP-8/L, or PDP-8/E CPU. For the PDP-12 there are two operating modes, parallel and serial.

In parallel mode, the FPP12 steals a maximum of 50 percent, or every other memory cycle, from the PDP-12, thus permitting the PDP-12 and the FPP12 to operate simultaneously. Once initiated in serial mode, the FPP12 locks out the PDP-12 CPU for the duration of a complete calculation. Serial mode increases the FPP12 calculation speed by approximately 20 percent.

The FPP12 performs arithmetic operations on floating-point numbers 20 to 100 times faster and with 100 to 200 fewer memory cycles than software interpreters. The FPP12 instruction set facilitates the programming of complicated algorithms and the building of compilers for mathematical languages. Variable length instructions are part of a flexible addressing scheme. Direct addressing of 32K of

core memory is available using a 24-bit instruction format. A 12-bit instruction format, in which the operand address is relative to a programmable base register, reduces program length and facilitates re-entrant coding. Any eight sequential core locations can be used as an index register to modify operand addresses. Index registers are adjusted prior to use in address modification, to account for the different number of core locations used in the two data formats permitted by the FPP12.

1.3 FLOATING POINT NUMBER SYSTEM

The term floating point implies a movable binary point similar to the movable decimal point used in scientific notation. An exponent is used to keep track of the number of spaces the binary or decimal point is moved.

Examples of scientific notation:

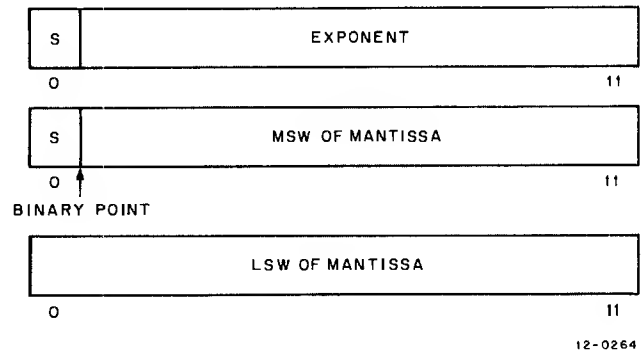
$$234 = 23.4 \times 10^1 = 2.34 \times 10^2$$

Examples of binary floating-point notation:

$$(1011) = (101.1) \times 2^1 = (10.11) \times 2^2 = (1.011) \times 2^3$$
$$(1.011) \times 2^3 = 0.1011 \times 2^4 = 0.01011 \times 2^5$$

In the example of binary floating-point notation given above, there are four significant bits. However, in the last term, the fraction that multiplies the exponent contains six bits. Given a fixed number of bits, it is desirable to adjust the exponent and the binary point to eliminate leading zeros. This adjustment retains the maximum numerical significance for a given format length. The FPP12 normalizes or removes leading zeros as the last step in every floating-point arithmetic operation.

The floating-point data format used by the FPP12 is identical to the format used by the PDP-8 floating-point system (DEC-08-YQYB-D). As shown below, there is a 12-bit signed 2's complement exponent and a 24-bit signed 2's complement mantissa.

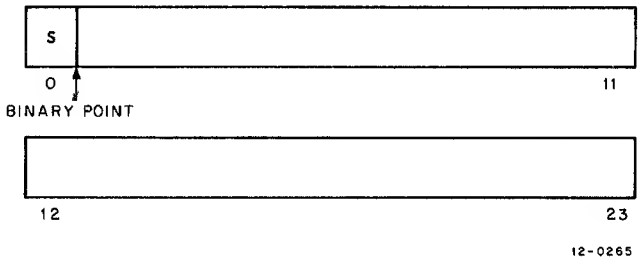


The FPP12 carries all calculations to 28 bits of precision, then rounds to 24 bits after normalization; after rounding, the result is rechecked for proper normalization prior to completing the instruction.

In fixed-point arithmetic, the precision of a number varies with the number's magnitude. In addition, the range of fixed-point numbers is generally limited in most mini computers to 12 or 16 bits. With the FPP12, the number range is $_2{}^{+2047}$ to $_2{}^{-2048}$; precision is maintained at 24 bits throughout the number range. Exceeding the upper limit, $_2{}^{+2047}$, causes the FPP12 to interrupt the PDP-12 CPU and set its exponent overflow status bit. A calculation resulting in a exponent smaller than $_2{}^{-2048}$ is an exponent underflow that can cause a program interrupt. At initialization, the programmer has the option to request that the underflow trap be ignored, in which case the result of calculation in which underflow occurred is set to 0.

Fixed-Point 24-Bit Fraction Format

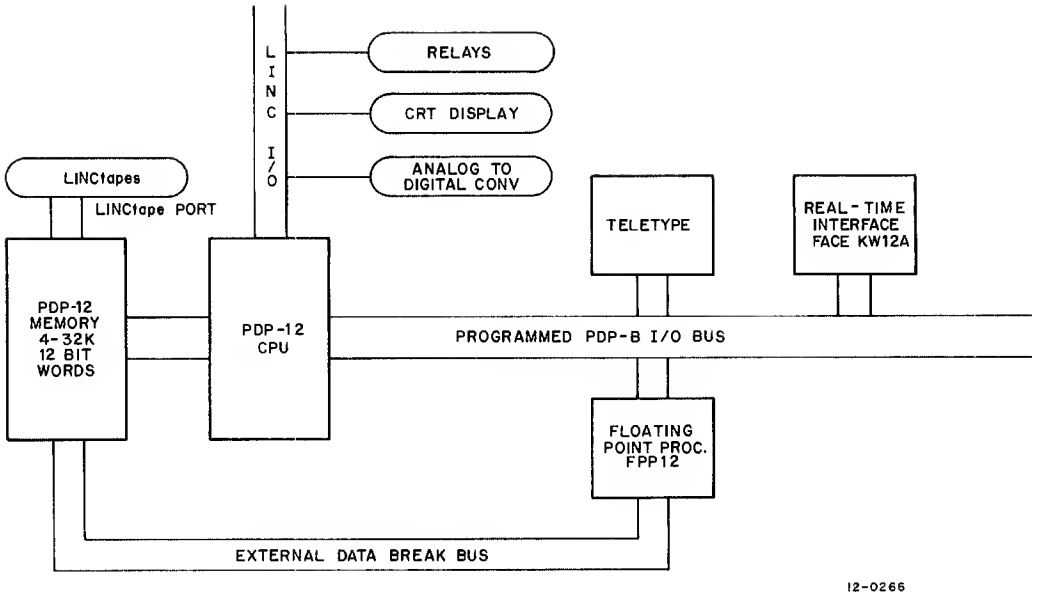
For those calculations where full 24-bit precision is not necessary and where core space is at a premium, the FPP12 can be used in fixed-point 24-bit mode. Each operand consists of a 24-bit signed 2's complement fraction as shown below.



As in the case of the floating-point mode, each calculation is carried to 28 bits of precision and rounded to 24 bits but no normalization is performed. Therefore, leading zeros occur which reduce the precision of subsequent calculations. In fixed-point mode, calculations resulting in a fraction overflow cause the FPP12 to initiate a program interrupt with the fraction overflow status bit set to 1.

1.4 SYSTEM DESCRIPTION

The FPP12 is initialized and interrogated as to its status via PDP-8 IOT instructions issued on the programmed I/O bus. Once initialized, the FPP12 operates as a processor fetching instructions and operands via the data break or direct memory access bus. A typical system configuration consisting of a PDP-12 and a FPP12 is shown in Figure 1-1. Note that the PDP-12 Computer contains two data break ports; one is permanently reserved for the LINCtape control, the other is available for a device



The FPP12 attaches to the EXTERNAL data break and programmed I/O bus of the PDP-12 computer without additional hardware.

Figure 1-1 PDP-12 12/40 System Configuration

such as a disk, data break card reader control CD12, or the FPP12. If two or more data break devices, in addition to the LINCtape, are attached to the PDP-12, a memory multiplexer (DM12) is generally required (see Figure 1-2). The exception is that the Analytical Instrumentation Package (AIP12) and the FPP12 may both be attached to the data break bus without a memory multiplexer. A special cable that connects the FPP12 to the AIP12 contains the signals necessary to arbitrate data break requests between the AIP12 and the FPP12.

On the PDP-8, PDP-8/I, and PDP-8/L Computers only one direct memory access port is available; attaching an FPP12 and DECtape, for example, requires a memory multiplexer. Each data break device has its own memory port on the PDP-8/E Computer; therefore, a separate memory multiplexer is not required on the PDP-8/E for up to 12 separate data break devices.

1.5 FPP12 REGISTER ORGANIZATION

There are eight registers in the FPP that are of interest to the programmer. The functions of these registers, named below, will be discussed in the remainder of this chapter.

Register	Function
Floating Point Accumulator (FAC)	36-bit register split into 12-bit exponent and 24-bit fraction.
Index Register Address Pointer (X0)	Contains the 15-bit core location of index register 0.

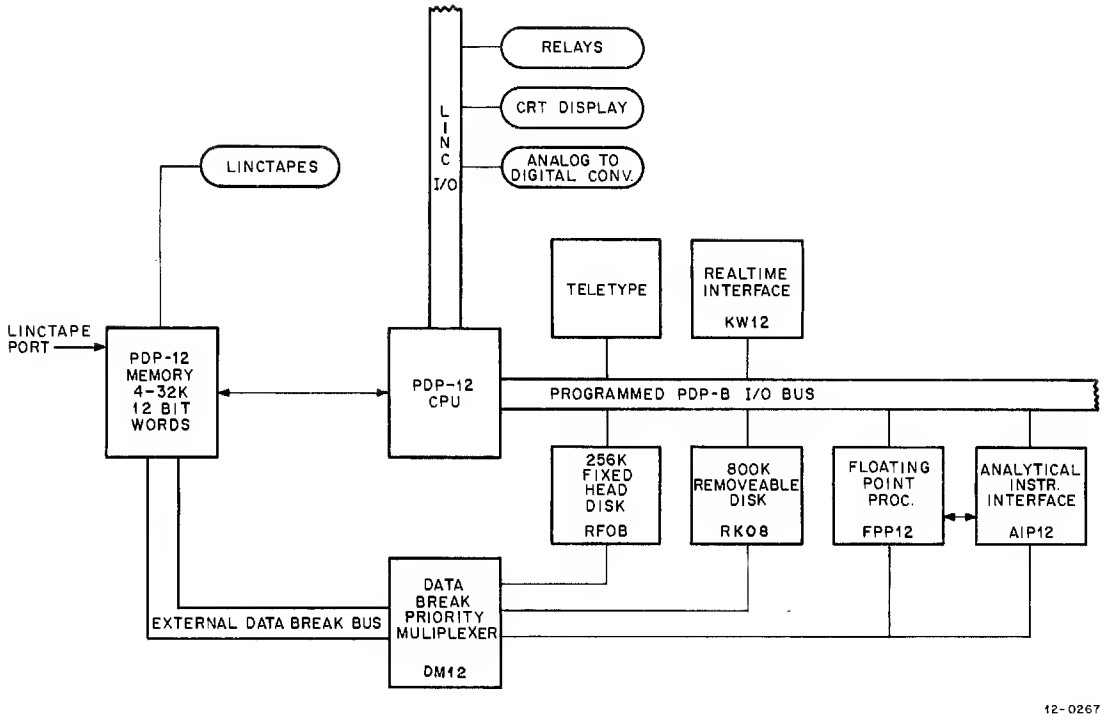


Figure 1-2 Typical Configuration of the PDP-12 with Multiple Devices

Register	Function
Base Register (Base)	Contains the 15-bit base address used in calculating single-word addresses.
Floating Point Program Counter (FPC)	Contains the 15-bit address that is the location of the next FPP12 instruction.
Active Parameter Table Pointer	Loaded via IOTs with the 15-bit address of the first location of the Active Parameter Table (APT).
Command Register	The command register is loaded with an IOT instruction. The command register selects FPP12 operating modes, sets the FPP12 interrupt enable, chooses the important parameters to be saved in the APT, and fixes the most significant 3 bits of the 15-bit APT pointer.
Status Register	The status register may be interrogated by the CPU to determine the cause of an exit operation by the FPP12. The status register also indicates if the FPP12 is in the run or (run) Λ (FPAUSE) state.

Register	Function
Operand Address Register	The operand address register is deposited in the APT and contains one of the following: a. If the last address-bearing instruction prior to the exit was of the data reference class, the operand address register contains the 15-bit address of the least significant word of the operand. b. If the last address-bearing instruction prior to the exit was an executed jump instruction, the operand address register contains the jump address. c. If after initialization an exit is performed prior to the execution of a jump or data reference instruction, the operand address register contains the FPC originally set by the APT. d. The instructions SET BASE (SET B) and SET X0 REGISTER (SET X) have no effect on the operand address register.

1.6 ACTIVE PARAMETER TABLE

The Active Parameter Table (APT) (refer to Table 1-1) contains information necessary for starting or restarting an FPP12 program. The APT is defined as any two to eight locations in core. The APT pointer is set to point at the first entry of the APT. The initialization procedure for the FPP12 includes two IOT instructions that set up a command register and set the 15-bit APT pointer to the first location of the APT, shown as location P in Table 1-1. Following the second IOT, the FPP12 picks up the contents of the APT. Whenever the FPP12 performs an EXIT, the current contents of the APT overlay the initial APT contents.

The APT performs three services for the programmer.

a. It reduces the number of IOTs necessary to initialize the FPP12. This reduces the CPU program overhead which is critical in multitask and time-shared environments.

b. It automatically saves the status of interrupted FPP12 programs.

c. It provides convenient access to the information necessary for debugging FPP12 programs and determining the cause of FPP12 "error" exits such as exponent overflow, underflow, or attempted division by 0.

With the exception of the operand address, all parameters contained in the APT are used in initializing the FPP12. The operand address is stored for the use of the CPU program when the FPP12 exits.

Table 1-1
Active Parameter Table Format

Location	Contents			
P	Field Bits of Operand Address	Field Bits of Base Register	Field Bits of Index Register Location	Field Bits of FPC
P+1	Lower 12 bits of FPC			
P+2	Lower 12 bits of Index Register 0 location X 0			
P+3	Lower 12 bits of Base Register			
P+4	Lower 12 bits of operand address			
P+5	Exponent of FAC			
P+6	MSW of FAC			
P+7	LSW of FAC			
NOTE: APT address points to location P.				

1.7 PROGRAMMING THE FPP12

The FPP12 is initialized and interrogated by PDP-8 type IOT instructions. Once started, the FPP12 operates much like an actual processor, fetching instructions and operands and storing results in the PDP-8 or PDP-12 core memory. Data breaks or "stolen" memory cycles are generally requested by the FPP12 as needed. The maximum number of breaks requested is generally one per regular PDP-8 or PDP-12 instruction. This means that while the FPP12 is operating, PDP-8 or PDP-12 programs can be run simultaneously at 50 to 70 percent of normal speed. Typically LINCtape, display, analog data acquisition, and other forms of I/O can be performed by the PDP-12 Computer while the FPP12 is calculating.

An optional mode is available to the FPP12 attached to a PDP-12 Computer. For calculations where the maximum FPP12 program speed is required, setting the proper command register bit (refer to Table 1-2) locks out the PDP-12 processor during FPP12 program execution. Using the "lock out" mode on the PDP-12 speeds up FPP12 programs by 15 to 25 percent (refer to Table 1-4).

1.7.1 Initialization

To execute the first instruction of any program, the FPP12 must have the 15-bit core address of the first instruction that is contained in the first two locations of the APT. The contents of other locations

Table 1-2
AC After Read Status Instruction 6556

AC Bit	Function if AC Bit Set to 1
0	Fixed-point mode.
1	Trapped instruction caused exit.
2	FPHLT instruction caused exit.
3	Attempted divide by 0 caused exit. The FAC was not altered.
4	Fraction overflow in fixed-point mode caused exit.
5	Exponent overflow caused exit.
6	Exponent underflow has occurred. Exit on exponent underflow is optional.
7, 8, 9	Unused
10	The FPP12 is currently paused.
11	The FPP12 is currently in a run state.

Table 1-3
Command Register Setting 6533

AC Bit	Function when AC Bit Set to 1
AC bits 0-11 have the following function when the FPCOM IOT is issued.	
0	Select fixed-point mode upon initiation.
1	Exit if exponent underflow occurs. Otherwise, set result of calculation and continue.
2	Forbid access to 4K memory fields other than the field that is occupied by the last location of the APT.
3	Enable CPU program interrupt when FPP12 Interrupt Request flag is set. Skip is always enabled.
4	Do not store operand address on exits. The operand address is never retrieved on initiate.
5	Do not store the address of index register 0 from or in the APT.
6	Do not store the base register from or in the APT.
7	Do not store the FAC from or in the APT.
8	Lock out the PDP-12 processor during FPP12 program execution. Unused on PDP-8 FPP12 systems.

(Continued on next page)

Table 1-3 (Cont)
Command Register Setting

AC Bit	Function when AC Bit Set to 1
AC bits 0-11 have the following function when the FPCOM IOT is issued.	
9 10 11	Most-significant 3 bits of APT pointer.
Note: Setting bits 4-7 of the command register speeds up initiation and exit operations. Setting command register bits 4-7 does not alter the relative position of items on the APT. In multijob environments, command register bits 4-7 are typically set to 0.	

Table 1-4
Instruction Execution Times*

Instruction	Octal Code	Serial Mode		Parallel Mode**	
		Double Precision Execution Time (μs)	Floating Point Execution Time (μs)	Double Precision Execution Time (μs)	Floating Point Execution Time (μs)
FLDA	0200+X	12	13	14	16
FADD	1200+X	13	19	14	23
FSUB	2200+X	13	19	14	23
FDIV	3200+X	24	28	27	31
FMUL	4200+X	23	27	27	31
FADDM	5200+X	17	26	24	30
FMULM	6200+X	27	33	30	39
FSTA	7200+X	12	13	14	16
*All times were measured using the single-word direct reference format. Timing tolerance is ±20%.					
**For these measurements the PDP-12 was performing mostly single cycle instructions.					

of the APT are often useful in starting a program and essential in restarting an interrupt task. Once the appropriate parameters are placed in an APT table by the CPU, two IOTs must be issued. FPCOM (6553) loads a command register and the most significant 3 bits of the APT pointer. The significance of the bits in the command register is shown in Table 1-3. FPST (6555) loads the least significant 12 bits of the APT pointer and starts the FPP12. A typical initiate sequence is shown in Example 2-1. Once initiated, the FPP12 will execute instructions until:

a. An error condition, such as exponent overflow, occurs.

b. An FEXIT instruction is encountered.

c. An FPHLT IOT is issued by this CPU.

d. An I/O preset is issued by the CPU.

e. The CPU encounters any type of halt.

1.7.2 Serial vs Parallel Processing

The most efficient use of resources occurs when the CPU and FPP12 are programmed to operate in parallel. For instance, in the display oriented research analysis (DORA) program which faciliates display interactive manipulation of data files, the PDP-12 refreshes a CRT display, performs Teletype®, LINCtape, and disk I/O, and samples knob and sense switch positions while the FPP12 is performing floating-point arithmetic. Because the FPP12 and the CPU access the same core memory, the communication methods are virtually unlimited; either processor can alter the other's program or data. Usually the CPU is assigned the job of scheduling and I/O, while the FPP12 performs complex arithmetic. However, in the DORA program, the FPP12 schedules I/O by passing parameters to the PDP-12 CPU.

There are occasions when it is desirable to complete an FPP12 calculation between operations performed by the CPU. Setting the appropriate command register bit in the FPP12 permits serial operation with the PDP-12 Processor. In serial mode, the PDP-12 CPU is locked out from the executing instructions while the FPP12 is operating.

There is no provision for a true serial mode for an FPP12 on a PDP-8 type processor. The fastest wait loop for a PDP-8, PDP-8/I, or PDP-8/L Computer consists of a JMP instruction with the programmed interrupt facility enabled, because data breaks can occur only between complete instructions. On the PDP-8/E Computer, the data break facility is structured so that data breaks may occur after any major state or multistate instructions. Therefore, the particular CPU program in progress does not affect the FPP12 instruction execution time on a PDP-8/E Computer.

1.7.3 IOT Instructions

A complete list follows of IOT instructions with device code 55 that apply to programming the FPP12. IOT instructions with device code 56 are relegated to maintenance programs. The use of maintenance IOTs is presented in Chapter 3. If a conflict exists between the FPP device select codes and the device select codes of another peripheral, the conflict must be resolved in the hardware by altering wired connections in either the FPP12 or the conflicting device. It is recommended that the FPP12 device codes not be altered because of the necessity of changing extensive diagnostic and system software. However, the logic to be altered in changing device codes is found on Prints FPP12-0-CI1, FPP12-0-CI2, and FPP12-0-CI3.

® Teletype is a registered trademark of Teletype Corporation.

1.7.4 IOT List

Mnemonic	Octal Code	Function
FPINT	6551	Skip when the FPP12 Interrupt Request flag is set.
FPICL	6552	Unconditionally reset the FPP12 including all flags. To the FPP12, the IOT FPICL is the same as I/O preset.
FPCOM	6553	If the FPP12 is not in a Run state and the FPP12 Interrupt Request flag is not set, the FPP12 command register is loaded with the contents of the AC*. If the FPP12 is in a Run state, or if the Interrupt Request flag is set, the FPCOM instruction is ignored.
FPHLT	6554	Force the FPP12 to exit, dump its status in the APT, and set the Interrupt Request flag at the end of the current instruction. The FPHLT instruction is used to abort an FPP12 program in a multijob environment or in software debugging. The following special features apply to the FPHLT instruction. a. If FPHLT is issued prior to the FPST instruction, the FPP12 will execute only one instruction after initiation and then exit with the FPC pointing to the succeeding instruction. This facilitates single stepping through an FPP12 program under CPU control. b. If the FPP12 is in a Pause state, the FPP12 will exit with the FPC pointing at the pause instruction. This means that if a job was aborted in a Pause state it will be resumed in a Pause state. c. Normally, if an exit is forced by FPHLT, AC02 will be set to a 1 when either read status FPRST or FPIST is issued. However, if the forced exit causes the FPP12 program to abort while an FEXIT instruction is being executed, the CPU forced exit flag is cleared. Thus, the CPU forced exit flag is an absolute indicator that a program was prematurely aborted.
FPST	6555	If the FPP12 is not running and the Interrupt Request flag is not set, the least significant 12 bits of the APT pointer are set to the contents of the AC and the FPP is started. If the FPP12 is in a Run state, but paused, the FPST instruction will cause the FPP12 to continue. Otherwise, the FPST instruction has no affect on the FPP12. If the FPST instruction causes the FPP12 to start or continue, the CPU will skip the instruction following FPST.
FPRST	6556	Read the FPP12 status register into the AC. FPRST may be issued at anytime.
FPIST	6557	Skip if the FPP12 Interrupt Request flag is set. If the skip occurs, read the FPP12 status register in the AC and clear the status flags and the Interrupt Request.

*AC refers to the PDP-12 or PDP-8 accumulator while FAC refers to the FPP12 accumulator.

1.7.5 Instruction Set-Address Methods

Three types of data reference instructions are available:

a. 24-bit instruction with a 15-bit absolute address.
b. 12-bit instruction with a 7-bit relative address.
c. 12-bit instruction with a 3-bit relative address that specifies a 15-bit indirect address.

Full indexing capability is available for types a and c. The determined operand address points at the exponent of the operand in floating-point mode and at the most significant word of the operand in fixed-point mode.

1.7.6 Index Registers

Any core location may be used as an index register. The core address of the current index register 0 is stored in the X0 register. The X0 register is initially set from the APT, but may be altered by the SET X instruction. Index register X is in location X0+X, where X = 0,....,7.

Accessing an array of data points requires incrementing the address of the current point by the data length to yield the address of each successive point. Index registers are used to accomplish this address modification. The index register is incremented by one to access each successive point, but it is multiplied by the data length, three for floating point and two for fixed point. This quantity is used as the displacement from the address specified in the instruction to yield the address of the current point. Adjusting of index registers simplifies "skipping" through data arrays and permits a single index register to be used as both a loop counter and address modifier (see Example 2-2). Pre-incrementing is selected by bit 5 of data reference instructions types a and c. Instructions are available for setting, testing, and performing arithmetic on index registers. In particular, the instruction ADDX, which adds the contents of bits 12-23 of the instruction to the contents of the index register specified by bits 9-11 of this instruction, is useful in manipulating "push-down stacks".

1.7.7 Instruction Set

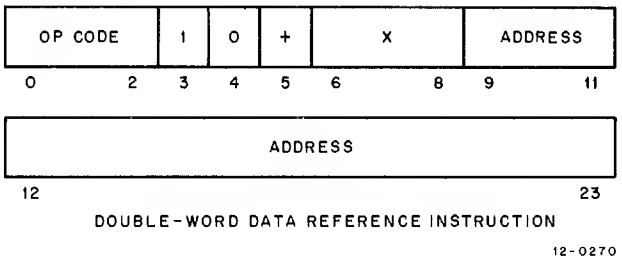
The FPP12 instruction set is divided into two basic classes: data reference instructions and special instructions. Data reference instructions are those that operate on the two data formats specified in Paragraph 1.8. Data reference instructions include the basic arithmetic operations plus load and store. All other instructions are special instructions that include index registers modifiers, jumps, pointer moves, and the operate-type instructions.

The instruction set is presented in detail in the following paragraphs. The instruction format follows each group of instructions.

1.8 DATA REFERENCE INSTRUCTIONS*

OP Code	Mnemonic	Data Function
0	FLDA	C(Y) → FAC
1	FADD	C(Y) + C(FAC) → FAC
5	FADDM	C(Y) + C(FAC) → Y
2	FSUB	C(FAC) - C(Y) → C FAC
3	FDIV	C(FAC)/C(Y) → FAC
4	FMUL	C(FAC) * C(Y) → FAC
7	FMULM	C(FAC) * C(Y) → Y
6	FSTA	C(FAC) → C Y

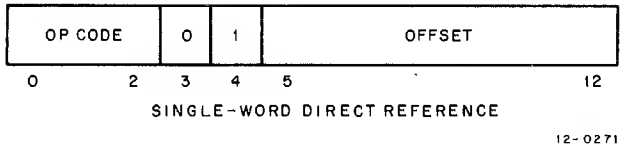
Data Reference Instruction Formats



DOUBLE-WORD DATA REFERENCE INSTRUCTION

12-0270

$$Y = C(\text{bits } 9\text{-}23) + M * (C(X + X0) + C(\text{bit } 5)) * \delta(X)$$

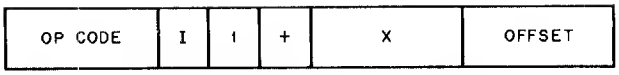


SINGLE-WORD DIRECT REFERENCE

12-0271

Y = C (base register) + 3 (offset)

*In fixed-point mode the exponent of the FAC is never altered.



SINGLE-WORD INDIRECT REFERENCE

12-0268

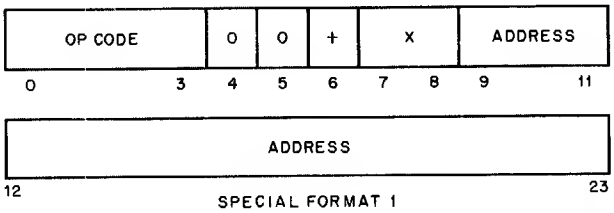
$$Y = C \text{ (bits 21-36 of } C(\text{(base reg.)} + 3^* \text{ offset)}) + (M) * (C\,X + X0) + C(\text{bit } 5))^* \delta(X)$$

$$\delta(X) = 1 \text{ if } X \neq 0 \text{ and } 0 \text{ if } X = 0$$

M = 2 if fixed-point mode
3 if floating-point mode

1.8.1 Special Format 1

OP Code	Mnemonic	Function
2	JXN	The index register X is incremented if bit 5 = 1 and a jump is executed to the address contained in bits 9-23, if index register X is nonzero.
3, 4, 5, 6, 7	Trapped Instructions	The instruction-trap status bit is set and the FPP12 exits causing a PDP interrupt. The unindexed operand address is dumped into the APT.



SPECIAL FORMAT 1

12-0269

1.8.2 Special Format 2

OP Code	Extension	Mnemonic	Function
0	10	LDX	The contents of the index register specified by bits 9-11 are replaced by the contents of bits 12-23.
0	11	ADDX	The contents of bits 12-23 are added to the index register specified by bits 9-11.

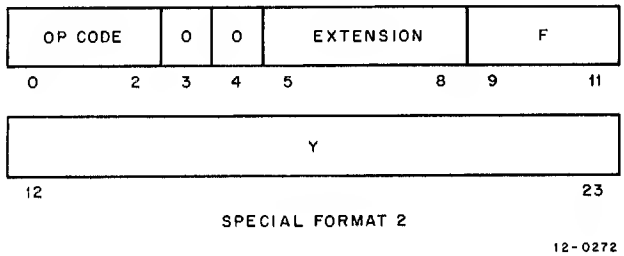
1.8.3 Conditional Jumps

Jumps, if performed, are to the location specified by bits 9-23 of the instruction.

OP Code	Extension	Mnemonic	Function
1	0	JEQ	Jump if FAC = 0
1	1	JGE	Jump if FAC $\geq$ 0
1	2	JLE	Jump if FAC $\leq$ 0
1	3	JA	Jump always
1	4	JNE	Jump if FAC $\neq$ 0
1	5	JLT	Jump if FAC < 0
1	6	JGT	Jump if FAC > 0
1	7	JAL	Jump if impossible to fix the floating-point number contained in the FAC; i.e., if the exponent is greater than $(23)_{10}$.

1.8.4 Pointer Moves

OP Code	Extension	Mnemonic	Function
1	10	SETX	Set X0 to the address contained in bits 9-23 of the instruction.
1	11	SETB	Set the base register to the address contained in bits 9-23.
1	13	JSR	Jump and save return. Jump to the location specified in bits 9-23 and the return is saved in bits 21-35 of the first entry of the data block.
1	12	JSA	An unconditional jump is deposited in the address and address + 1, where address is specified by bits 9-23. The FPC is set to address + 2.



SPECIAL FORMAT 2

12-0272

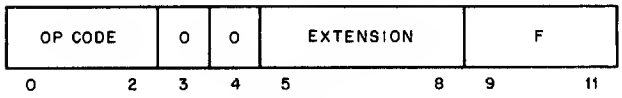
1.8.5 Special Format 3

OP Code	Extension	Mnemonic	Function
0	1	ALN	The mantissa of the FAC is shifted until the FAC exponent equals the contents of the index register specified by bits 9-11. If bits 9-11 are zero, the FAC is aligned so that the exponent = $(23)_{10}$.* In double-precision mode, an arithmetic shift is performed on the FAC fraction. The number of shifts is equal to the absolute value of the contents of the specified index register. The direction of shift depends on the sign of the index register contents. A positive sign indicates a shift toward the least significant bit, while a negative sign indicates a shift toward the most significant bit. The FAC exponent is not altered by the ALN instruction in double-precision mode.
0	2	ATX	The contents of the FAC are fixed and the least significant 12 bits of the mantissa are loaded into the index register specified by bits 9-11. In double-precision mode the least significant 12 bits of the FAC are loaded into the specified index register. The FAC itself is not altered by the FLATX instruction.
0	3	XTA	The contents of the index register specified by bits 9-11 are loaded right-justified into the FAC mantissa. The FAC exponent is loaded with $(23)_{10}$ and then the FAC is normalized. This operation is typically termed floating a 12-bit number. In double-precision mode, the FAC is not normalized.
0	4	NOP	The single-word instruction performs no operation.
0	5-7	reserved	These codes are reserved for instruction set expansion and should not be used.
0	12-17		
1	14-17		

*Setting the exponent = $(23)_{10}$ integerizes or fixes the floating-point number. The JAL instruction tests to see if fixing is possible.

1.8.6 Operate Group - Special Format 3

OP Code	Extension	9-11 Bits	Mnemonic	Function
0	0	0	FEXIT	Dump active registers into the APT, reset the FPP12 RUN flip-flop to the 0 state, and interrupt the PDP-12 processor.
0	0	1	FPAUSE	Wait for synchronizing signal. IOT FFST (6555) will restart the instruction following FPAUSE.
0	0	2	FCLA	Zero the FAC mantissa and exponent.
0	0	3	FNEG	Complement FAC mantissa.
0	0	4	FNORM	Normalize the FAC. In double-precision mode FNORM is a NOP.
0	0	5	START F	Start floating-point mode.
0	0	6	START D	Start double-precision mode.
0	0	7	JAC	Jump to the location specified by the least significant 15 bits of the FAC mantissa.



SPECIAL FORMAT 3

12-0273

CHAPTER 2
FPP12 PROGRAMMING EXAMPLES

2.1 INTRODUCTION

Programming examples for the Floating Point Processor and a procedure for initializing the FPP12 are contained in this chapter. Several examples are provided that utilize index registers. A re-entrant sine subroutine illustrates a technique for writing re-entrant code. Program debugging techniques are discussed in detail. The mnemonics and syntax used in this chapter are consistent with those of the FPP assembler. A complete description of the assembler can be found in the manual entitled, FPP Assembler Manual, DEC-12-AQZA-D. A math package for the FPP is described in a manual entitled FPP Support Library (DEC-12-YEXA-D).

2.2 PROGRAM INITIALIZATION

Each FPP12 program consists of one or more instructions and an Active Parameter Table (APT). Upon initialization, the APT (refer to Table 1-1) contains the initial setting of important FPP12 registers. Whenever the FPP12 finishes or aborts a program, the APT is updated before the CPU is interrupted.

The CPU program in Example 2-1 starts the FPP12 with the APT pointer set to location 01000, which is word 1000 of field 0. The FPP12 normally does not recognize page or field boundaries. If the APT started in location 07777, the least significant 12 bits of the FPC would be found in location 10000.

In Example 2-1, the FPP12 will pick up locations 02000, 02001, 02002, 02003, 02005, 02006, and 02007 of the APT. Note that the operand address, location 02004 in this example, is never retrieved from the APT by the FPP12. After retrieving the contents of location 02007, the FPP12 will fetch its first instruction from location 01000. The 4 in the second digit of the contents of location 01000 indicates that the instruction is a 2-word, direct addressing, data reference instruction. The 0 in the first digit of location 01000 indicates that the instruction is an FLDA. Bits 9-23 of the instruction specify the address, which is not indexed when bits 6-8 are all zero. After fetching the address, the FPP12 will break to 12000, 12001, and 12002 to load the operand into the FAC.

After retrieving the least significant word of the FAC from location 12002, the FPP12 will fetch another instruction from location 01002. The instruction in location 01002 is an FEXIT, equivalent

```
/ Sample program to initialize FPP12
                    ORG     00020   /Psuedo OP sets assembler origin at
                                      location (20)8 of field 0.
00020     2000      APTPT,  APT     /Pointer to APT

                    ORG 200
                    BEGIN,  CLA             /Clear AC
                            FPCOM           /Load 0's to FPP12 command register
                            TAD APTPT
                            FPST            /Set APT points to 02000 and start
                            HLT             /If no skip FPP12 is not ready to be started
                            FPINT           /Wait
                            JMP. -1
                            HLT             /Program done

/ A Sample
/ FPP12 Program is below

Loc             contents                    ORG 01000

01000           0401                        FLDA TAG    /Load contents of
01001           2000                                    /Location TAG into FAC
01002           0000                        FEXIT       /Dump APT
                                                        /Into core and interrupt CPU
/ Active parameter table

Loc             contents                    ORG 02000

02000           0                       APT, 0000       /most sig bits
02001           1000                         1000       /FPC
02002           3000                         3000       /X0
                                             4000       /Base
                                             ----       /Operand address
                                             ----       /FAC exp
                                             ----       /FAC MSW
                                             ----       /FAC LSW
ORG 12000
12000           0002    TAG,    3.0                     /constant (3.0)10
12001           3000
12002           0000
```

Example 2-1 Sample FPP12 Program

to a halt instruction for the CPU. Prior to stopping, the FPP12 dumps the current APT over the initial APT, beginning with the least significant word of the FAC in location 02007 and ending with location 02000. The APT at the completion of the FEXIT instruction is shown in Table 2-1.

Table 2-1
APT After FEXIT in Example 2-1

02000	1000	/current Field Bits
02001	1003	/current FPC
02002	2000	/X0
02003	4000	/Base
02004	2002	/Operand address
02005	0002	/exponent
02006	3000	/MSW
02007	0000	/LSW

Only after dumping the APT is the FPP12 Skip or Interrupt flag set. In Example 2-1, the CPU executes a WAIT loop while the FPP12 is operational. It would be far more efficient for the CPU to perform some other task, such as tape or Teletype I/O, while the FPP12 is calculating.

2.3 INDEX REGISTERS AS ADDRESS MODIFIERS AND LOOP COUNTERS

The FPP12 program in Example 2-2 moves a list of $(200)_8$ floating point numbers from an area of core starting at location ALPHA to an area starting at location BETA. Note that index registers are used both for loop counting and address modification. Index register 1 is set to -1 and index register 0 is set to -200 using the LDX instruction. Index register 1 is incremented prior to use as an address modifier for the FLDA instruction at location LOOP. Index register 0 is used as a loop counter by the JXN instruction.

```
BEGIN,      LDX   -1, 1          /set index register 1 = -1
            LDX   -200, 0        /set index register 0 = (-200)8
LOOP,       FLDA  ALPHA, 1+      /first C(1 + X0) = C (1 + X0)
                                  +1 Then load FAC from loc.
                                  ALPHA + C (1 + X0) *3
            FSTA  BETA, 1        /Store FAC in loc BETA + C
                                  (1 + X0) *3
            JXN LOOP, 0 +        /first C (X0 + 0) = C(X0 + 0)
                                  +1
                                 /then go to loop if C (X0 + 0) ± 0
            FEXIT                /trap to CPU

            ORG 4000
ALPHA,      --------
            ORG 6000
BETA,         0
```

Example 2-2 Move List from ALPHA to BETA Using Index Registers

It is possible to use the same index register as a loop counter and as an address modifier, because of the method used in the FPP12 hardware to calculate indexed addresses. In the process of formulating an address, the FPP12 checks to see if indexing is required. If indexing is required, the contents

af the specified index register are retrieved and "adjusted" by the appropriate multiplier, which is 3 far flaating-paint mode and 2 for fixed-point mode. Then the adjusted index register is added to the unindexed address and the resulting addition, initially performed with 24 bits of precision, is truncated ta 15 bits by dropping the 9 most significant bits of the result. Example 2-3 illustrates the standard method of indexed address calculation. If it is necessary to use index register 5 as a loop counter, additianal care must be used in selecting the pointer to list A contained in the instruction. Consider the case where the loap counter is set to $(-200)_8$. Then the pointer to list A must be modified to be $[A + M(C(I) + (30000)_8]$. C (I) is the initial setting of the index register and M is the number of 12-bit bytes in the data word. Example 2-4 is similar to Example 2-2; however, only ane index register is used.

Initially X0 = 14003
C(X0 + 5) = 0001

Instruction FLDA A, 5 0451
2003

A is location 12003

Address calculation proceeds as follows:

1. The contents of (X 0 + 5) are retrieved and multiplied by three.
2. The "adjusted" index register is added to 00012003 the unindexed address to yield 00012006.
3. This address is truncated to 12006.

Example 2-3 Indexed Address Calculation

2.4 USE OF INDEX REGISTERS TO CREATE PUSH-DOWN STACKS

The subroutines in Example 2-5 illustrate the use of the ADDX instruction in creating push-down or last-in-first-out lists. The PUSH routine is called with an argument in the accumulator. The POP routine returns with elements removed from the stack in the accumulator. These subroutines are designed to be called with the JSA instruction which places an unconditional jump to the return in the first two locations of the subroutine.

The PUSH and POP subroutines in Example 2-5 are valid for either fixed-point or floating-point mode, as lang as second and additional calls are in the same mode as the very first call.

```
BEGIN,      LDX COUNT, 1
LOOP,       FLDA ALPHA-(M*COUNT-50000), 1
            JXN LOOP, 1 + FSTA ALPHA-(M*COUNT-50000), 1
```

M = 3, if floating point mode
    2, if fixed point mode

$K = (70000)_8$

Example 2-4 Index Register 1 is Used as Both an Address Modifier and Counter

```
PUSH,       0
            0
            FSTA STACK, 2+      /Place contents of AC in stack
            JA PUSH             /Return from subroutine
POP,        0
            0
            FLDA STACK, 2       /Retrieve item from stack
            ADDX 2, -1          /Decrement stack pointer
            JA POP              /Return from subroutine
```

Example 2-5 Push-Down Stacks

2.5 BRANCH OR JUMP ON CONDITION INSTRUCTIONS

Seven conditional jump instructions are provided in addition to the JXN instruction. Six of these, JEQ, JGE, JLE, JNE, JCT, and JGT, test the FAC mantissa. The seventh, JAL, executes a jump if the FAC cannot be represented as a $(24)_{10}$ bit binary number. This occurs when the FAC exponent is greater than $(23)_{10}$ or $(27)_8$.

2.6 WRITING RE-ENTRANT SUBROUTINES

A re-entrant subroutine is one in which the code is not altered during execution. This property permits the interruption of a task which is executing a given re-entrant subroutine and the starting of another task that uses the same subroutine. The advantage of re-entrant coding is that two or more jobs can use the same subroutine without concern as to when a given job is interrupted.

The single-word data reference instructions and a re-entrant jump to subroutine facilitate the writing of re-entrant codes. With the JSR instruction, the return address is saved in bits 21-35 of the location pointed at by the contents of the base register. If it is necessary to store temporary values during subroutine execution, single-word instructions should be used. This will force addressing to be relative to the base register setting. Each task will have a unique base register setting; therefore, the effective addresses for temporary storage for each task will be unique, even though the offsets for the data instructions are never charged in the pure subroutine. The return from the re-entrant subroutine consists

of the two instruction sequence, FLDA ALPHA, JAC, shown in Example 2-6. JSR causes the return address to be deposited into the first location of the data block, ALPHA, which is defined by the base register. The return address is deposited into the FAC with the instruction FLDA ALPHA. The JAC instruction actually executes the return jump by setting FPC equal to bits 9-23 of the FAC.

```
/ Main Prog
MPROG,   JSR SUB          /Jump to sub prog.
         FEXIT
SUB,     FLDA ALPHA       /Load return address
         JAC              /Jump to the address contained in
                          /bits 9-23 of the FAC fraction
         Base ALPHA
ALPHA,     ---
```

Example 2-6 Return from Re-Entrant Subroutine

2.7 USE OF THE FPHLT INSTRUCTION

The FPHLT IOT (6554) permits the CPU to force the abortion of a running FPP12 program or to force the FPP12 to execute one instruction each time it is initialized. In a multitask or time-shared environment, it is often necessary to suspend a calculation prior to completion. When debugging a program, it is often desirable to examine the results of each instruction's execution.

If FPHLT is issued while the FPP12 is executing a program, that program will be aborted at the end of the current FPP12 instruction. The FPP12 will dump the current APT in core and then cause a CPU program interrupt. If the current instruction is anything except FEXIT, status bit 02 will be set to 1 if FPHLT forced the FPP12 to stop program execution.

Issuing FPHLT prior to FPST will cause the FPP to initialize, execute one instruction, then exit. By repeating this procedure, the CPU can force the FPP12 to single step through a program.

2.8 DEBUGGING FPP12 PROGRAMS ON UNITS ATTACHED TO PDP-12 COMPUTERS

The PDP-12 console (described in the PDP-12 System Reference Manual) is a powerful tool for debugging FPP12 programs. Using the switches, one can single step through FPP12 programs, observing the transfers between the FPP and the PDP-12 memory on the console lights. Alternatively, the FPP12 program can run until a specific memory address is accessed, in which case the computer will halt, permitting the console light to be examined. While the computer is halted, memory may be examined

and altered with the switch register without disturbing the program counters associated with either the CPU or the FPP12. IOT instructions may be issued with the console switches that examine registers within the FPP12.

If the stop switch is raised during the execution of a FPP12 program, the PDP-12 will stop at the end of a complete instruction or a data break caused by some external device such as the FPP12. Depressing the continue switch with the stop switch raised causes the execution of one CPU instruction or one data break for each actuation of the continue switch. Operating in this mode, the FPP12 will receive one data break for each CPU instruction. This means that every other time the continue switch is depressed a data break will occur. Whenever the break indicator light is lit, the MA and MB lights on the console refer to the data break address and memory buffer contents associated with the FPP12 program. The single step switch causes similar results, except the halts occur at the end of each major state of the CPU instructions. The single step switch is useful when the CPU program that runs in parallel with the FPP12 program contains tape instructions. The stop switch has no affect for the duration of LINCtape instructions.

If bit 8 of the FPP12 command register is set to 1, the CPU will be locked out while FPP12 programs are executing. This is reflected in the fact that the break light will stay on continuously as the continue switch is actuated.

2.9 USING THE EXECUTE STOP SWITCH

If the execute stop switch is raised the PDP-12 will halt whenever the memory location whose address is contained in the left switches is accessed during any cycle except a CPU fetch cycle. Setting the left switches to the first location of the next APT to be used and raising the execute stop switch causes the PDP-12 to halt following the first FPP12 data break following FPST IOT.

2.10 CARE NECESSARY IN THE USE OF EXAMINE AND DEPOSIT SWITCHES

Some care is necessary when using the examine and deposit switches, if they are to be used while a FPP12 program is temporarily halted. Problems arise because of the logical implementation of the break field register within the PDP-12. The 4K memory field examined on the first push of the examine switch following a program may be the field into which the FPP12 was breaking when the program stop occurred. To be sure that the proper data for an examine operation is displayed in the MB register, the examine switch should be actuated twice for the first operation following a program stop. When the computer system is restarted, the first PDP-12 cycle following an examine or deposit operation will be a break cycle if the FPP12 is requesting a data break. To ensure that the FPP12 breaks to the proper 4K memory field, the last operation after any series of examines and deposits must be a fill step.

2.11 ADDITIONAL PROGRAMMING HINTS

2.11.1 Illegal Mantissa

In the 2's complement number system the number consisting of a one followed by twenty-three zeros is an illegal number because it and its 2's complement are both equal to -1. The FPP12 logic will not allow this number to be generated as the result of any calculation. For instance, if -1/2 is added to -1/2 the result shows up in the FPP as -1/2 *2 or -1. It is possible for this number to arise in other than calculations. For instance, it is possible to intentionally place a number into core memory from the CPU's switches. The routine in Example 2-7 illustrates a test for the illegal fraction.

```
/The value in location A possibly has an illegal fraction

BEGIN,    FLDA      A       /Get C (A)
          JGE       GOOD    /If C(A) 0 all is OK
          FNEG              /Form 2's complement of fraction
          JLT BAD
GOOD,     FEXIT             /Number is OK
BAD,      FEXIT             /Number has illegal fraction
```

Example 2-7 Test for Illegal Fraction 100000000...000

Example of Re-Entrant Sine and Exponential Subroutines

Examples 2-8 and 2-9 contain the FPP code for calculating SINE (X) and X (X **Y). The comments indicate what each step of the routines is doing. Both subroutines are written in the mnemonics and syntax of the FPP assembler.

```
0001
0002                / SINE USES THE 1ST 3 ENTRIES IN
0003                / THE BLOCK AND INDEX REG. 0,1&2
0004                / X IS PASSED THROUGH THE 2ND ENTRY
0005                / IN THE BLOCK AND SIN(X) IS RETURNED
0006                / THROUGH THE SAME LOCATION
0007                      ORG 10500
0010                      BASE 0
0011                      X=1*3
0012                      XSQR=2*3
0013                / CALCULATE ABSOLUTE VALUE OF X,INDEX
0014                / REG 0 SET TO 0 INDICATES SIGN OF X
0015                / WAS NEGATIVE
0016 10500 0201 SINE,     FLOA X
0017 10501 0100           LDX -1,0          /INITIATE INDEX REG 1
     10502 7777
0020 10503 1061           JGT CAL           /GO TO CAL IF X IS POSITIVE
     10504 0512
0021 10505 1001           JEQ DONE          /GO TO DONE IF X IS 0
     10506 0603
0022 10507 0003 MOO,      FNEG              /NEGATE FAC
0023 10510 0100           LDX 0,0           /SET INDEX REG TO ZERO
     10511 0000
0024                / REDUCE X TO 1ST CYCLE USING THE
0025                / IDENTITY SIN(X)=SIN(N*2*PI+X)
0026 10512 3401 CAL,      FDIV TWOPI        /DIVIDE X BY 2*PI
     10513 0607
0027 10514 6201           FSTA X
0030 10515 1071           JAL ERROR         /X IS TOO LARGE
     10516 0606
0031 10517 0010           ALN 0
0032 10520 0004           FNORM             /GET INTEGER PART
0033 10521 2201           FSUB X
0034 10522 0003           FNEG              /GET FRACTIONAL PART
0035 10523 1001           JEQ DONE          /SIN(2*PI*N) IS ZERO
     10524 0603
0036 10525 4401 REM,      FMUL TWOPI        /NORMALIZE TO BETWEEN 0 AND 2*PI
     10526 0607
0037 10527 6201           FSTA X
0040                / REDUCE X TO 1ST HALF CYCLE USING
0041                / THE IDENTITY SIN(X)=-SIN(X-PI) FOR
0042                / PI<X<=2*PI
0043 10530 2401           FSUB PI
     10531 0612
0044 10532 1051           JLT PCHECK        /IF X IS LESS THAN PI GO TO PCHECK
     10533 0543
0045 10534 6201           FSTA X            /SET X TO X-PI
0046 10535 2101           JXN RESET,0+      /IF INDEX REG 0 WAS -1 SET TO 0 AND
     10536 0541
0047 10537 1031           JA PCHECK+1       /GO TO PCHECK+1
     10540 0544
0050 10541 0100 RESET,    LDX -1,0          /IF INDEX REG 0 WAS 0 SET IT TO -1
     10542 7777
```

Example 2-8 SINE Routine (Sheet 1 of 2)

```
0052                    / REDUCE X TO 1ST QUARTER CYCLE USING
0053                    / THE IDENTITY SIN(X)=SIN(PI-X) FOR
0054                    / PI/2<X<=PI
0055 10543 0201 PCHECK, FLDA X
0056 10544 2401         FSUB PIBY2      /IF X IS LESS THAN OR EQUAL TO PI/2
     10545 0615
0057 10546 1021         JLE PALG        /GO TO PALG
     10547 0555
0060 10550 0401         FLDA PI
     10551 0612
0061 10552 2201         FSUB X          /REPLACE X WITH PI-X
0062 10553 1031         JA PALG+1
     10554 0556
0063 10555 0201 PALG,   FLDA X
0064 10556 3401         FDIV PIBY2
     10557 0615
0065 10560 6201         FSTA X          /NORMALIZE X TO BETWEEN 0 & 1
0066 10561 4201         FMUL X
0067 10562 6202         FSTA XSQR       /CALCULATE X**2
0070 10563 0101         LDX -4,1        /SET UP INDEX REG 1
     10564 7774
0071 10565 0102         LDX -1,2        /SET UP INDEX REG 2
     10566 7777
0072 10567 0002         FCLA
0073                    / CALCULATE SIN(X)=((((C9*(2*X/PI)**2
0074                    / +C7)*(2*X/PI)**2+C5)*(2*X/PI)**2
0075                    / +C3)*(2*X/PI)**2+PI)*2*X/PI
0076 10570 1521 LOOP,   FADD C9,2+       /ADD C9 ON 1ST PASS, C7 ON
                                          2ND PASS, ECT.
     10571 0620
0077 10572 4202         FMUL XSQR       /MULTIPLY PARTIAL SUM BY X**2
0100 10573 2111         JXN LOOP,1+     /GO TO LOOP 4 TIMES
     10574 0570
0101 10575 1401         FADD PIBY2
     10576 0615
0102 10577 4201         FMUL X
0103 10600 2001         JXN DONE,0      /GO TO DONE IF X WAS POSITIVE
     10601 0603
0104 10602 0003         FNEG            /NEGATE ANSWER
0105 10603 6201 DONE,   FSTA X          /STORE ANSWER
0106 10604 0200         FLDA 0
0107 10605 0007         JAC             /RETURN TO CALL
0110 10606 0000 ERROR,  FEXIT           /EXIT ON ERROR
0111 10607 0003 TWOPI,  3.1415926*2.0
     10610 3110
     10611 3756
0112 10612 0002 PI,     3.1415926
     10613 3110
     10614 3756
0113 10615 0001 PIBY2,  3.1415926/2.0
     10616 3110
     10617 3756
0114 10620 7764 C9,     +1.5146190E-04
     10621 2366
     10622 5615
0115 10623 7771 C7,     -4.6737656E-03
     10624 5466
     10625 6317
0116 10626 7775 C5,     +7.9689679E-02
     10627 2431
     10630 5053
0117 10631 0000 C3,     -6.4596371E-01
     10632 5325
     10633 0420
```

Example 2-8 SINE Routine (Sheet 2 of 2)

```
0001                / EXP USES THE 1ST 6 ENTRIES IN
0002                / THE BLOCK
0003                / INDEX REG, Ø MUST BE SET TO THE
0004                / POSITION OF THE EXPONENT OF THE
0005                / 5TH ENTRY IN THE BLOCK
0006                / X IS PASSED THROUGH THE 2ND ENTRY
0007                / IN THE BLOCK AND EXP(X) IS RETURNED
0010                / THROUGH THE SAME LOCATION
0011                      ORG 10500Ø
0012                      BASE Ø
0013                      X=1*3
0014                      F=1*3
0015                      FSQR=2*3
0016                      TEMP=3*3
0017                      IDXØ=4*3
0020                / CALCULATE THE ABSOLUTE VALUE OF X
0021                / INDEX REG 3 SET TO Ø INDICATES THAT
0022                / THE SIGN OF X WAS NEGATIVE
0023 10500 0201 EXP,      FLDA X -          /GET X
0024 10501 0103           LDX -1,3          /INITIATE INDEX REG 3 TO -1
     10502 7777
0025 10503 1041           JNE NZRO          /GO TO NZRO IF X IS NOT EQUAL TO ZERO
     10504 0511
0026 10505 0401           FLDA K1           /SET FAC TO 1
     10506 0601
0027 10507 1031           JA RETURN         /RETURN TO CALL
     10510 0573
0030 10511 1061 NZRO,     JGT GTZERO        /GO TO GTZERO IF X WAS POSITIVE
     10512 0516
0031 10513 0103           LDX Ø,3           /SET INDEX REG 3 TO Ø TO INDICATE
                                             X WAS NEGATIVE
     10514 0000
0032 10515 0003           FNEG              /NEGATE THE FAC
0033 10516 4401 GTZERO, FMUL LG2E           /MULTIPLY X BY LOG2(E)
     10517 0576
0034 10520 6203           FSTA TEMP         /STORE RESULT TEMPORARILY
0035 10521 0401           FLDA K1
     10522 0601
0036 10523 6204           FSTA IDXØ         /SET IDXØ TO 1=2**1*1/2
0037 10524 0203           FLDA TEMP
0040 10525 0010           ALN Ø             /FAC=N=INTEGER PART OF X*LOG2(E)
0041 10526 0020           ATX Ø             /IDXØ=2**N*1/2
0042 10527 0110           ADDX 1,Ø          /IDXØ=2**(N+1)*1/2=2**N
     10530 0001
0043                / THE 5TH ENTRY IN THE BLOCK
0044                / CONTAINS 2**N WHERE N IS THE
0045                / INTEGER PART OF X*LOG2(E)
0046                / FIND F=FRACTIONAL PART OF X*LOG2(E)
0047 10531 0004           FNORM             /FAC CONTAINS INTEGER PART OF
                                             X*LOG2(E)
0050 10532 2203           FSUB TEMP
0051 10533 0003           FNEG              /FAC CONTAINS FRACTIONAL PART OF
                                             X*LOG2(E)
0052 10534 6201           FSTA F
```

Example 2-9 Exponential Subroutine (Sheet 1 of 2)

```
0054                    / CALCULATE 2**F=1+2*(A-F+B*F**2+
0055                    /              C/(D+F**2)
0056 10535 4201                 FMUL F
0057 10536 6202                 FSTA FSQR       /FSQR=F**2
0060 10537 1401                 FADD D
     10540 0620
0061 10541 6203                 FSTA TEMP       /TEMP=D+F**2
0062 10542 0401                 FLDA C
     10543 0615
0063 10544 3203                 FDIV TEMP       /FAC=C/(D+F**2)
0064 10545 2201                 FSUB F
0065 10546 1401                 FADD A
     10547 0607
0066 10550 6203                 FSTA TEMP       /TEMP=A-F+C/(D+F**2)
0067 10551 0401                 FLDA B
     10552 0612
0070 10553 4202                 FMUL FSQR
0071 10554 1203                 FADD TEMP
0072 10555 6203                 FSTA TEMP       /TEMP=B*F**2+A-F+C/(D+F**2)
0073 10556 0201                 FLDA X
0074 10557 4401                 FMUL K2         /FAC=2*F
     10560 0604
0075 10561 3203                 FDIV TEMP
0076 10562 1401                 FADD K1         /FAC=1+2*F/(B*F**2+A-F+C/(D+F**2))
     10563 0601
0077                    / CALCULATE EXP(X)=2**(X*LOG2(E))=
0100                    /                  (2**N)*(2**F)
0101 10564 4204                 FMUL IDX0
0102 10565 2031                 JXN RETURN,3    /GO TO RETURN IF X WAS POSITIVE
     10566 0573
0103                    / CALCULATE EXP(-X)=1/EXP(X)
0104 10567 6201                 FSTA X
0105 10570 0401                 FLDA K1
     10571 0601
0106 10572 3201                 FDIV X
0107 10573 6201 RETURN, FSTA X          /STORE RESULT IN X
0110 10574 0200                 FLDA 0
0111 10575 0007                 JAC             /RETURN TO CALL
0112 10576 0001 LG2E,   1.442695
     10577 2705
     10600 2434
0113 10601 0001 K1,     1.0
     10602 2000
     10603 0000
0114 10604 0002 K2,     2.0
     10605 2000
     10606 0000
0115 10607 0007 A,      9.954596E+01
     10610 3070
     10611 5703
0116 10612 7774 B,      3.465736E-02
     10613 2157
     10614 5161
0117 10615 0012 C,      6.179723E+02
     10616 2323
     10617 7434
0120 10620 0007 D,      8.741750E+01
     10621 2566
     10622 5341
```

Example 2-9 Exponential Subroutine (Sheet 2 of 2)

CHAPTER 3
FPP12 COMPONENTS

3.1 FPP12 HARDWARE DESCRIPTION

The FPP12 is a peripheral processor that attaches to both the programmed I/O bus and the data break I/O bus. Figure 1-2 shows a typical configuration of an FPP12 attached to a PDP-12, with several other peripherals. It is of major importance to fully understand the differences between the I/O bus of the PDP-12, LINC-8, PDP-8, PDP-8/I, PDP-8/L, and PDP-8/E Computers. All of DEC's 12-bit computers share a compatible I/O structure. Most peripherals such as the FPP12 are nearly plug-in compatible with all of these computers. Major differences are listed below:

a. PDP-8/L, PDP-8/E, PDP-12, and some PDP-8/I's have what is referred to as a positive I/O bus which implies that the I/O signal levels are TTL compatible. The PDP-8, LINC-8, and some PDP-8/I's have a negative I/O bus which implies that the I/O signal levels are 0 and -3V with reference to chassis ground. Bus driver and receiver modules in the FPP12 are selected for either positive or negative bus computers.

b. The sense of the IOP pulses is inverted on those computers with a negative I/O bus. To account for this, certain wiring changes must be made on FPP12 logic to convert it to negative bus units. These changes are detailed in Chapter 4. If the original purchase order for the FPP specifies an FPP12N, the negative bus changes will be factory installed.

c. Data break timing on the PDP-12 Computer differs slightly from data break timing on the PDP-8 type computers. On the PDP-12, the trailing edge of ADDRESS ACCEPT indicates memory buffer strobe; on the PDP-8s, PDP-8/I, PDP-8/L, and PDP-8/E the leading edge of BUFFERED TIME STATE 3 indicates memory buffer strobe. The line that carries the signal BUFFERED TIME STATE 3 on the PDP-8 type computer is the same one that carries BUFFERED TIME STATE 5 on the PDP-12 Computer. Therefore, the FPP12 wired in the positive-bus PDP-8/I, PDP-8/L, and PDP-8/E configuration will operate on a PDP-12 but will not achieve optimum performance.

d. Raising pin N16V2 of the I/O cable on a PDP-12 Computer will lock out the CPU. The FPP12P will utilize this option when command register bit 8 is set to 1. Time comparisons are shown in Table 1-3. On computers other than the PDP-12, pin N16V2 is used for different purposes. Therefore, run F05U2 - B03V2 is deleted in the FPP12 when it is configured for processors other than the PDP-12.

Data break timing diagrams for the PDP-12 and other Family of 8 Computers are shown in Figures 3-1 and 3-2.

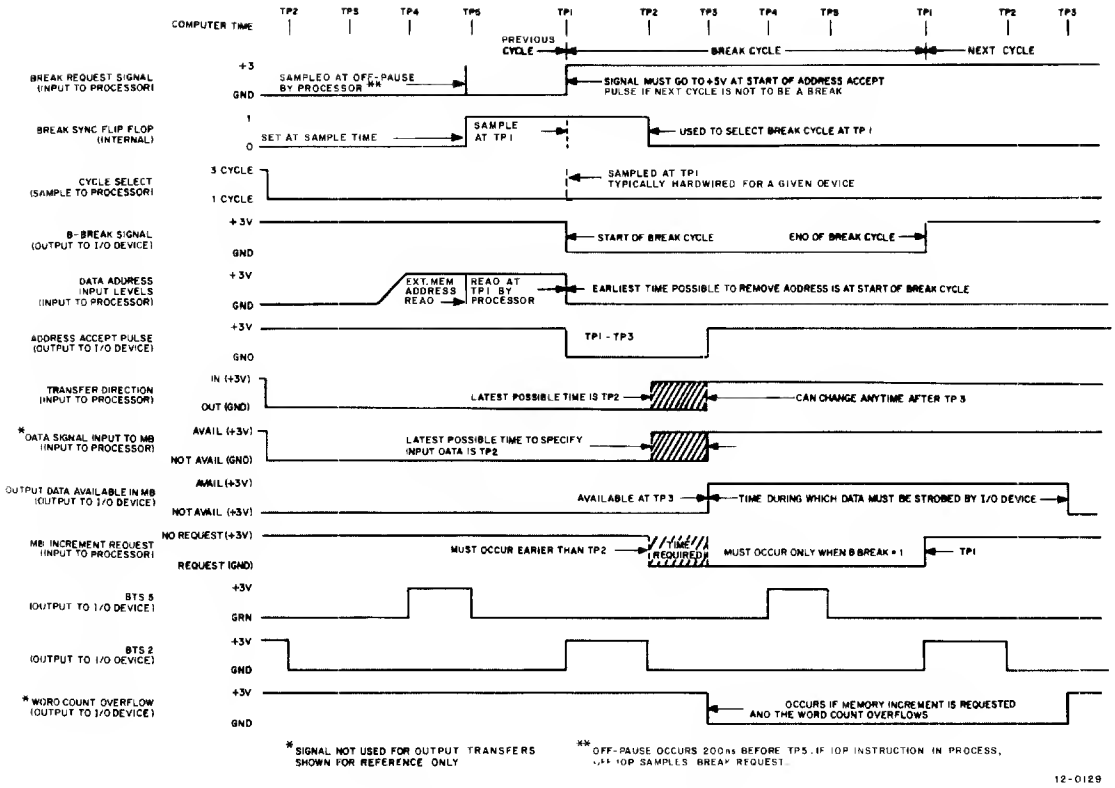
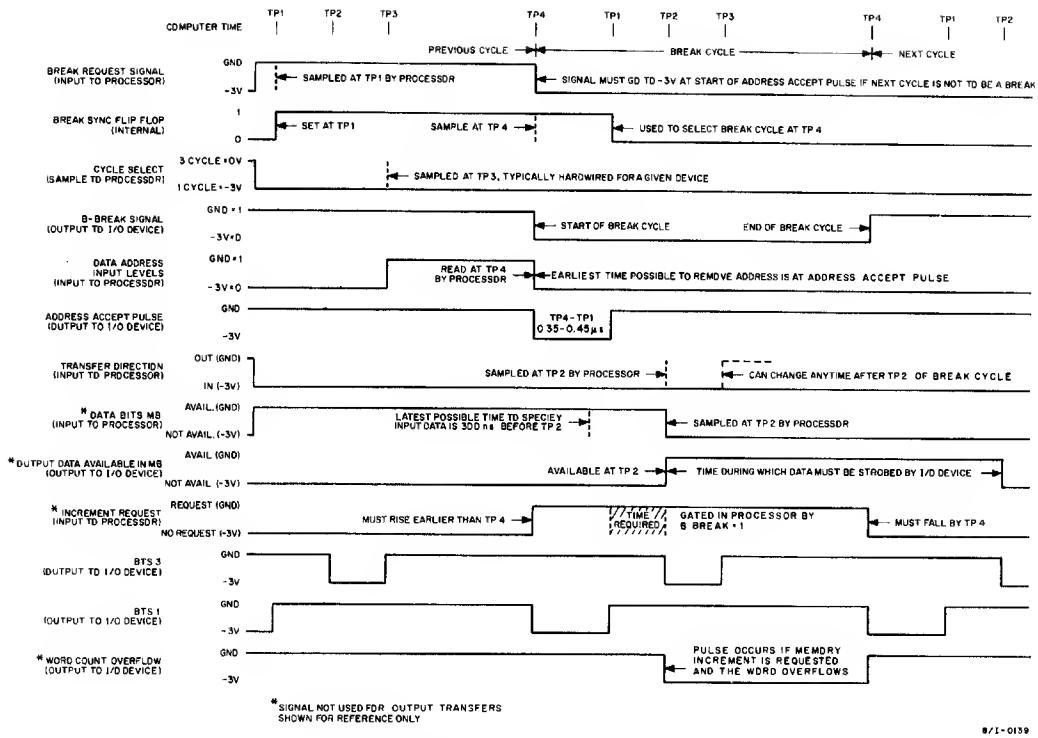


Figure 3-1 PDP-12 Single Cycle Data Break Timing

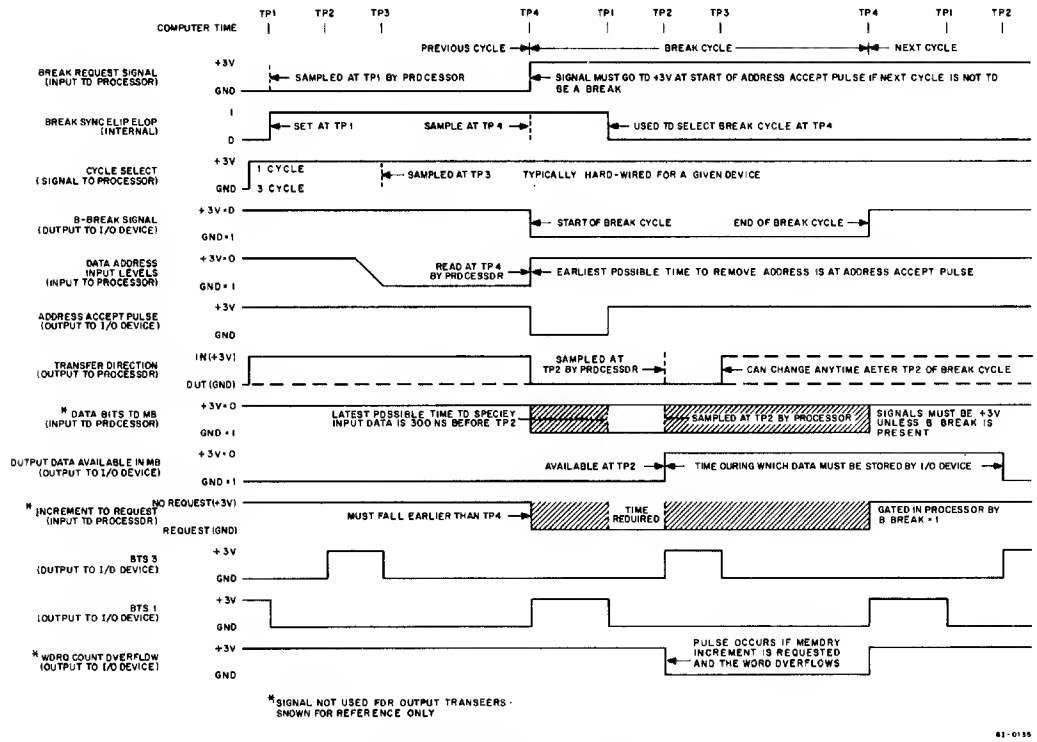
3.2 DESCRIPTION OF REGISTERS

Floating Point Processor Systems organization is shown in Figures 3-3, 3-4, and 3-5. The User IOT Decoder System (see Figure 3-3) describes the simplest communication path between the CPU and the FPP12. IOT (device code 55) instructions of interest to systems programmers are described in detail in Chapter 1. Maintenance IOT's (device code 56) are described in Paragraph 3.9.

The FPP12 Timing and Enable system are shown in Figure 3-4. The Major State and Time Slot generator provides up to 16 major time states in any 6 major or enable states. Each major time state contains 4 mini states; therefore, a total of 384 time slots are provided by the FPP12 timing system. The timing diagram for the state generator is shown in Figure 3-6. Typically at any one instant of time, one or two gates in one of the 6 state enable sections is qualified. A qualified gate in the state enable system may conditionally qualify any number of Register Gates. A conditionally qualified register gate causes a register transfer on the next clock pulse. It is appropriate to observe that the FPP12 logic is fully clocked, i.e., all flip-flops change state on the occurrence of a pulse from the system clock generated by a free-running RC oscillator adjusted to a frequency of 5 mHz.



Negative I/O Bus & Logic



Positive I/O Bus & Logic

Figure 3-2 PDP-8 Single Cycle Data Break Timing

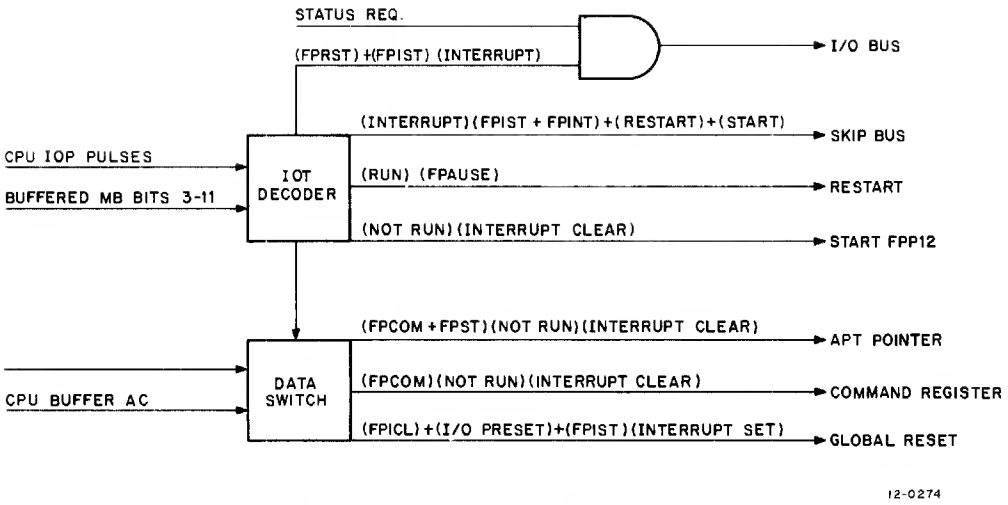


Figure 3-3 FPP12 User IOT Decoder System

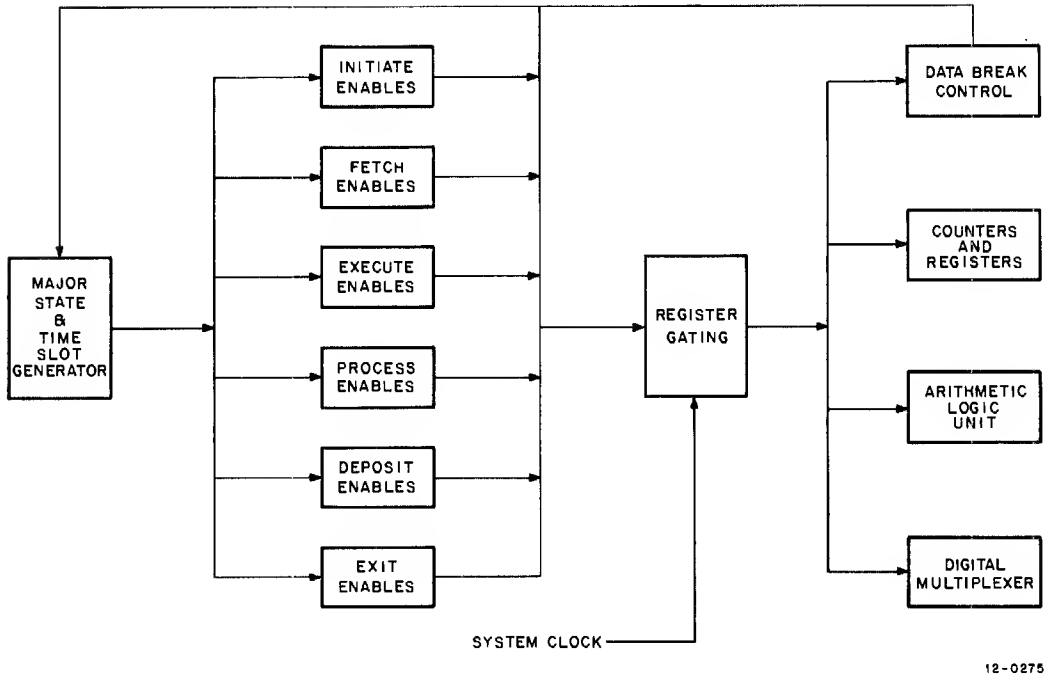


Figure 3-4 Timing and Enable System in FPP12

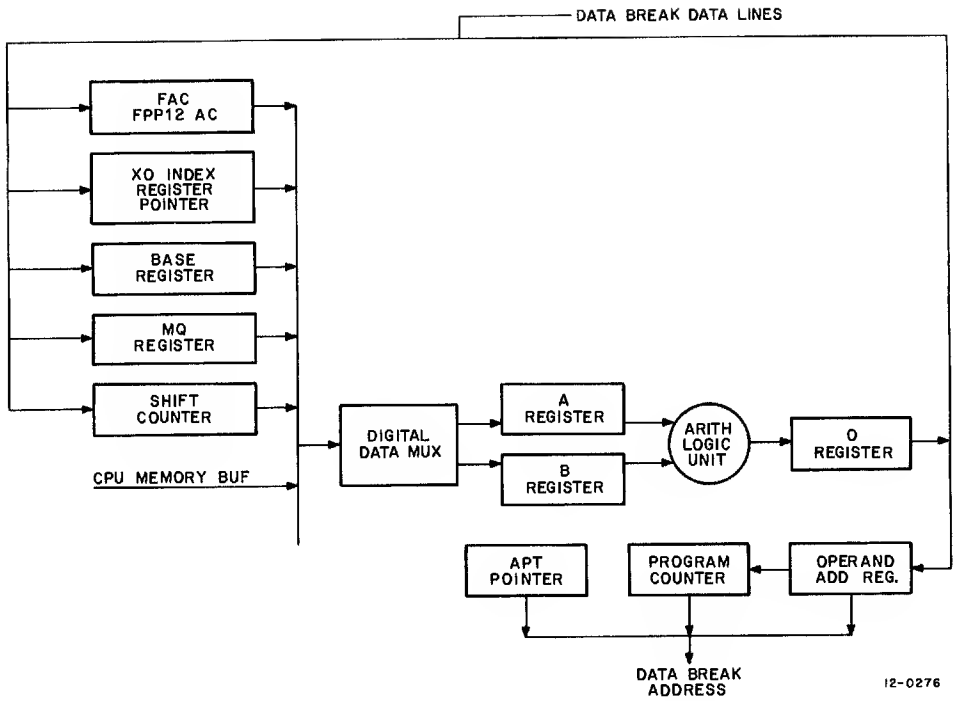


Figure 3-5 FPP12 Data Flow System

The FPP12 data flow system is shown in Figure 3-5. In some respects FPP12 architecture is similar to the PDP-8 in that major registers are multiplexed through a central arithmetic logic unit. However, FPP12 logic design is based on the use of medium-scale integrated circuit technology (MSI). The Operand Address Register, Program Counter (FPC), and APT pointer are formed from 4-bit binary up/down counters. This permits the incrementing of address registers and the performance of arithmetic operations on data variables simultaneously. The arithmetic logic unit consists of seven 24-pin MSI devices that can each perform all 16 Boolean and 16 different arithmetic functions on two variables. Full carry-look-ahead permits the addition or subtraction of two variables in under 100 ns.

3.3 MAJOR STATES

The FPP12 logic is organized into six major states:

INITIATE
FETCH
PROCESS
EXECUTE
DEPOSIT
EXIT

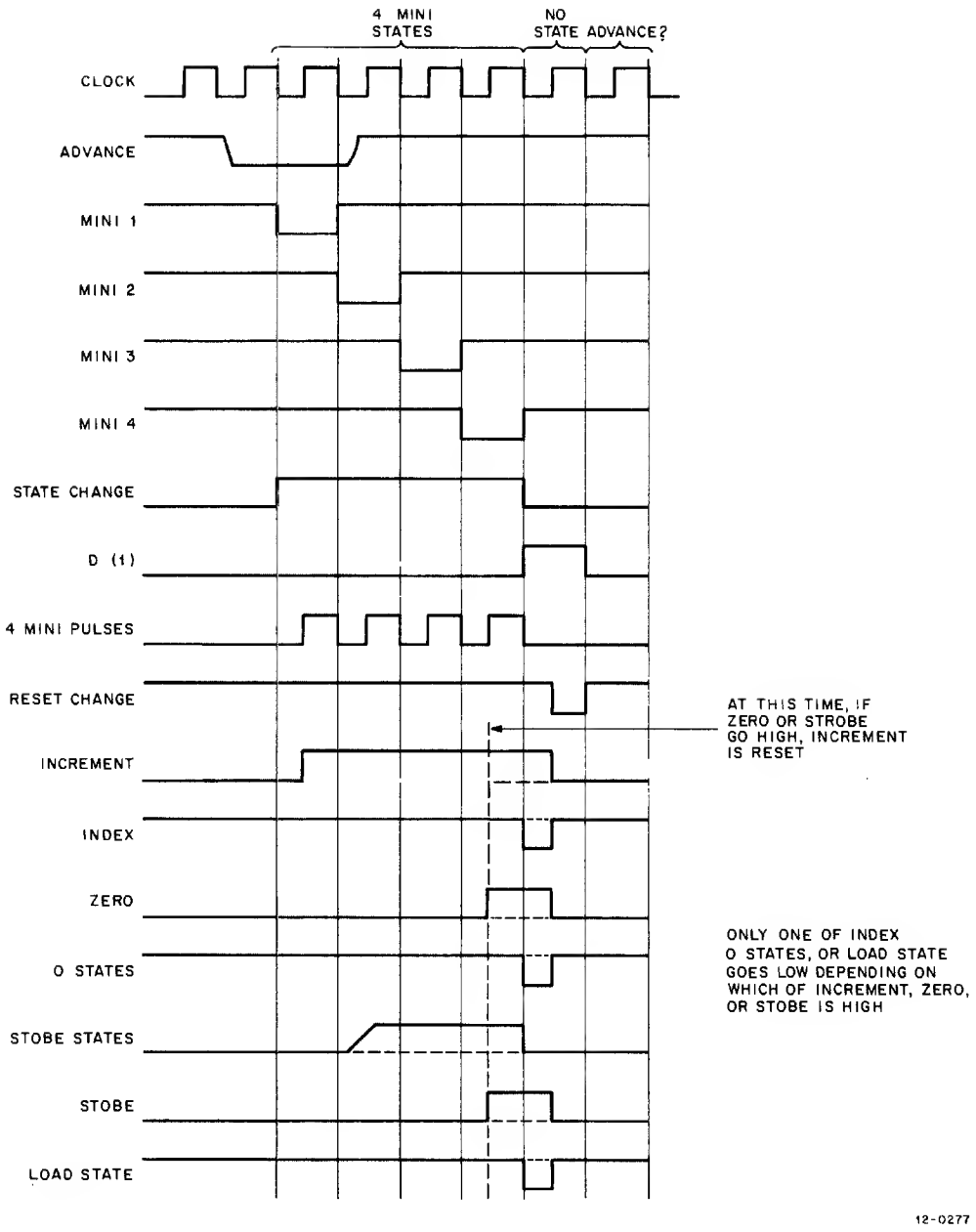


Figure 3-6 Timing Diagram of State Generator

With one exception, if any major state flip-flop on print CNR is set the FPP12 will be actively calculating. If all major state flip-flops are reset, the FPP12 will be inactive. The single exception has to do with the instruction FPAUSE, which causes the FPP12 to wait for a synchronizing signal before preceding.

The FPP12 operations that occur in each major state are detailed below:

INITIATE

INITIATE begins at the trailing edge of IOP 4 when IOT instruction 6555 is issued by the CPU, if the FPP12 is not running and the FPP12 and the FPP12 Interrupt Request flag is reset. During INITIATE, the contents of the APT are retrieved.

NOTE

Only the first two locations of the APT must be used as these contain the 15-bit initial setting FPP12 program counter (FPC). The fifth location of the APT, the operand address, is not retrieved during INITIATE.

Following the completion of INITIATE the FPP12 proceeds to FETCH time state 0.

Schematic drawings for INITIATE are found on print ARS2.

FETCH

During FETCH, preliminary decoding of instructions occurs. Instructions that require only one major time state to be completed, such as FCLA, are completely finished during FETCH State 0. Special instructions that require more than one major time state, such as ALN 0, or more than one memory cycle, such as ATX, cause the FPP12 logic to go from FETCH State 0 to PROCESS State 1. All data reference instructions require FETCH to continue beyond FETCH State 0 in order to calculate the operand address. At the end of the FETCH cycle for all data reference instructions, a transfer is made to EXECUTE State 0, with the Operand Address Register appropriately loaded. FETCH begins in State 0 and ends when the address calculation is complete. FETCH schematics are found on prints FTH1 through FTH3.

PROCESS

Most special instructions that require more than one major time state or more than one memory cycle are completed in PROCESS. With the exception of NORM and XTA, the FPP12 returns to FETCH State 0 after the completion of PROCESS. Processing for XTA and NORM is completed in DEPOSIT. PROCESS begins in major time state 1. PROCESS schematics are found on prints SPI1 through SPI3.

EXECUTE

The execution of all data reference instructions begins during EXECUTE. FLDA and FSTA are completed during EXECUTE. For all other data reference instructions the FPP12 proceeds to DEPOSIT at the completion of EXECUTE if no EXECUTE error is encountered. EXECUTE errors are defined as:

a. An attempt to divide by 0

b. A fraction overflow in fixed-point mode.

EXECUTE (Cont)

At the completion of EXECUTE for instructions other thon FLDA and FSTA, the un-normolized result of any colculotion is stored in the O register and an exponent is stored in the MQLSW. The exponent contained in the MQLSW is the operand exponent for FMUL, FMULM, ond FDIV ond the resultant exponent before normalization for FADD, FADDM, and FSUB. The shift counter is 0 if no froction overflow occurred, ond 1 if o floating-point froction overflow occurred. EXECUTE schematics are found on prints AST0 through AST3.

DEPOSIT

DEPOSIT begins in major time stote 11 ond ends in major time stote 15. As DEPOSIT is the only function performed during these time stotes, DEPOSIT enables shown on prints DEP1 through DEP3 are not gated with the DEPOSIT flip-flop found on the CNR print. During DEPOSIT the following functions are performed in the order listed:

a. The results of oll floating-point arithmetic calculations are normolized. The number of shifts is stored in the shift counter.

b. The normalized result is rounded to 24 bits.

c. For FMUL ond FMULM the FAC exponent is odded to the MQLSW.

d. For the FDIV instruction the MQLSW is subtrocted from the FAC exponent.

e. The contents of the shift counter ore added to the un-normalized exponent.

f. If the exponent resulting after normalization is within bounds, -2048 to +2047, the resultant answer is stored in the FAC for all operotions except FADDM ond FMULM. For FADDM and FMULM, the resultant answer is stored in the addressed location. After storing the resultant onswer, the FPP12 returns to FETCH State 0, unless the IOT FPHLT was issued by the CPU during the current instruction.

g. If the exponent is not within bounds ofter normolizotion the appropriate stotus bit is set ond the FPP12 enters EXIT State 0. DEPOSIT schemotics ore found on prints DEP1 through DEP3.

EXIT

During EXIT the current APT is deposited into core over the initiol APT. Only the first two locations of the APT must be deposited; the other locotions ore optionol occording to the commond register setting. The items in the APT are olwoys located in the some position relotive to one another. If the progrommer chooses not to deposit the operond oddress the fifth locotion of the APT is simply skipped. The field bits of the base register, X0, ond FPC are the first retrieved on INITIATE ond lost deposited during EXIT.

EXIT is entered for ony of the following conditions:

a. A FEXIT instruction is encountered.

b. A froction overflow occurs in fixed-point mode.

c. An exponent overflow or underflow occurs in flooting-point mode. If EXIT is entered for on exponent underflow commond, register bit 1 is tested. If it is set to 1, the EXIT is continued. If it is set to 0, the result of the previous colculotion is set to 0

EXIT (Cont)

and the FPP12 returns to FETCH State 0. If an exponent underflow occurs, status bit 6 is set as an indicator, even if command register bit 1 is set to 0.

d. An attempt to divide by 0 is made.

e. A FPHLT IOT is issued by the CPU.

At the end of EXIT the FPP12 halts in major time state 0 with all major state flip-flops reset. The FPP12 skip flag is set and the CPU program interrupt is actuated if command register bit 3 is set to 1.

3.4 DESCRIPTION OF REGISTERS

Most major registers of the FPP12 are described in Paragraph 1.5. The MQ, shift counter, A register, B register, and O register are hidden from the programmer. Their characteristics are as follows:

MQ — The MQ is a 28-bit parallel-serial input, parallel output, left-right shift register. It is divided into an MQLSW of 12 bits and an MQMSW of 16 bits. During multiplication, the MQ contains the absolute value of the multiplier mantissa. During division, the MQ temporarily contains the un-normalized absolute value of the quotient. When entering deposit, the MQLSW contains the un-normalized resultant exponent for FADD, FADDM, and DSUB and the operand exponent for FMUL, FMULM, and FDIV. The MQ is found on print MQR1.

SHIFT COUNTER — The SHIFT COUNTER is a 6-bit binary up/down counter. As its name implies, it is used to count shifts. The SHIFT COUNTER is found on print CAR 6.

A REGISTER AND B REGISTER — These 28-bit registers are inputs to the arithmetic logic unit (ALU) of the M190. The A register is a 28-bit storage register; the B register is a parallel-serial input parallel output shift register that shifts towards the least significant bit.

O REGISTER — The O register is the 28-bit output buffer for the M190. It is a parallel-serial input, parallel output shift register that shifts towards the most significant bit.

The A, B, and O Registers are found on the AMSW, ALSW, and EXT prints.

The following registers (listed below with the prints on which they are found) were described in Paragraph 1.5.

FAC	CAR 2 and CAR 3
X0 REGISTER	CAR 4
BASE REGISTER	CAR 5
FPC	CAR 1
APT POINTER	CAR 1
OPERAND ADDRESS REGISTER	CAR 1
STATUS REGISTER	CAR 8
COMMAND REGISTER	CAR 8

The operand address register, the FPC, and the APT pointer provide the addresses for data breaks. These registers are attached to a digital multiplexer that drives the EXT address lines of the CPU. The FAC, X0 register, base register, 0 register, MQ, and shift counter feed the digital multiplexer that loads the A and B Registers.

3.5 REGISTER GATING SYSTEM

The OR gates in the register gating system found on prints RG1 through RG10 funnel enables from many sources into signals that, when added with a clock pulse, cause a register action. This action can be a load, shift, count, or a clear. The signals that actuate the data multiplexer are found on prints RG7 through RG10. The multiplexer gates are enabled for at least the duration of a mini time state. The time from the beginning of the mini time state until the clock pulse, shown on Figure 3-6, is allowed for the data to settle on the register inputs.

3.6 DATA BREAK CONTROL

The FPP12 accesses core memory via the single-cycle data break facility. The data break control serves the following functions:

a. Channels data break and direction requests to the CPU from the state enables.

b. Gates the proper address register onto the EXT ADD bus of the CPU.

c. Gates the proper data register onto the EXT DATA bus of the CPU if an input break is required.

d. Synchronizes the FPP12 time state generator to the particular CPU memory timing to which the peripheral is attached.

The synchronizing logic for the data break control is shown on print DBC1. The signal DBC1 REQUEST BREAK L requests the data break from the processor. This signal is actuated by the condition:

(REQ BRK CYCLE (1) H) (BREAK (0) H)

The first term is the output of a flip-flop that permits the FPP12 to remember that it is currently requesting a data break. The second term is the signal from the CPU that a break is not in progress. Once the break cycle begins, noted by the disqualification of Break (0) H, the FPP12 break request must be removed.

In the lower right-hand side of the DBC1 prints there is a signal DBC1 ENAB DATA H. The equation for this signal is:

(BREAK (1)H) (DBC1 REQ BRK CYCLE (1) H) + CI2 MAINT READ L

The signal DBC1 ENAB DATA H permits the placing of data on the I/O bus during the break cycle requested by the FPP12. The DONE flip-flop, which is clocked by the trailing edge of ADDRESS ACCEPT in the PDP-12 or the falling edge of BTS3 in the PDP-8, restarts the FPP12 timing chain.

A data break may be initiated by any of the function enables placing a low level on the input of three sets of OR gates found on DBC2. These OR gates funnel break requests to the DBC1 REQ BRK CYCLE flip-flop and choose which of three address registers to use for the break address.

A data break for the purpose of retrieving data from core memory is an OUTPUT BREAK. A data break for the purpose of storing data in core memory is an INPUT BREAK. If a core memory location is incremented an INCREMENT BREAK is performed. The FPP12 data source for INPUT BREAKS is selected on DBC3.

There are four data sources used for INPUT BREAKS. They are: the field bits of the APT, the operand address register, the least significant word of the B multiplexer, and the most significant word of the A multiplexer.

If the data source selected for an INPUT BREAK is either the A or B multiplexer, an additional eight possibilities exist. The actual data source is resolved on prints RG7 through RG10.

There are several wiring changes specified for converting a positive bus FPP12 logic to a negative bus logic. These changes are in Chapter 4 of this manual.

3.7 MODULES INTRODUCED IN THE FPP12

There were three etch boards and five new modules introduced for the FPP12. The following list shows the module number and the function of these modules.

Module No.	Function
M155	One of 16 decoders using 74154 IC decoders.
M190	4-bit arithmetic logic module using 74181 arithmetic logic unit integrated circuit.
M191	Two carry look-ahead 74182 ICs for the 74181.
M238	Two separate 4-bit synchronous binary up/down counters with separate up and down clocks. Uses two 74193 ICs.
M245	Two separate, 4-bit parallel-serial input, parallel serial output shift registers. Uses two 8271 ICs.

The M178 Digital Multiplexer, with 8 inputs for each of 6 outputs, is used in the FPP12 although this module was originally built for a new PDP-10 processor. The M191, M238, and M245 use a common etch board, the 5008912, which is a mount for 2-16 pin dip packages with pin 16 reserved for +5V and pin 8 reserved for ground. The M155 is constructed on a 24-pin DIP mount, the 5008908. The M190 is a unique module layout containing 8 ICs including the 24-pin arithmetic logic unit.

Full specifications for new MSIs may be found in either the DEC specification file or from the manufacturer's catalog. The 74182, 74181, 74193, and 74154 ICs are listed in catalog number CC301 by Texas Instruments Inc. The 8271 and the 8291 (74197 from TI) are listed in Signatic's MSI Specifications Handbook, DCL Vol. II.

All five of the modules introduced with the FPP12 are tested on Digital Equipment Corporation's computerized module tester.

3.8 FLOW DIAGRAMS

The flow diagrams are the key to troubleshooting the FPP12 logic. In order to understand the flow diagrams, it is necessary to understand the timing generator and the register structure. It is recommended that the reader thoroughly study Chapter 3, Paragraphs 3.1-3.7 and Chapter 4 of this manual before attempting to understand this section.

A brief review of the timing generator and the register structure is presented here. The timing generator has the following properties:

1. There are 16 possible major time states. These are named State 0 through State 15.
2. The timing generator can be forced to jump to any state; that is, if the timing generator is in State 3, the next state could be State 11 if the proper enables are generated.
3. During the time a state is enabled, four different enabling pulses are generated; they are called Mini State 1 through Mini State 4. These pulses occur sequentially and are one clock cycle long. The end of the major time state occurs at the end of Mini State 4 and the next major time state is enabled at the trailing edge of the next clock cycle. When a state is entered, the timing generator stops until it receives a timing advance. When the advance is enabled, the four mini states are generated. During an output break cycle, Mini State 1 is used to enable the clocking of the data from the MB into the appropriate register.

The characteristics of the M190 are important also in understanding the flow diagrams. Recall that there are three registers; the outputs of the A and the B registers are connected to the two inputs of the ALU, DEC 74181. The output of the ALU is connected to the input of the O register. The A, B, and O registers each have unique properties. The A register can be loaded with the true or the complemented value of the inputs. The B register is a shift register that can be shifted toward the least

significant bit; the O register is a shift register that can be shifted toward the most significant bit. It is important to realize the many functions that the DEC74181 can perform. The functions used in the FPP12 are listed in Table 3-1, along with the enables that are required to perform the function.

Table 3-1
Functions Performed by DEC74181

	S_3	S_2	S_1	S_0	Inhibit Carry	Carry In
A minus B → O	L	H	H	L	L	H
A plus B → O	H	L	L	H	L	L
A plus A → O	H	H	L	L	L	L
A → O	H	H	H	H	L	L
A plus 1 → O	H	H	H	H	L	H
Logical 1 → O	L	L	H	H	H	-
B → O	H	L	H	L	H	-
Logical 0 → O	H	H	L	L	H	

Note that the function B minus A cannot be performed. In order to do this the 1's complement must be loaded in A, A plus B with a carry insert enabled, and the result loaded into O.

The following paragraphs contain a brief description of each of the flows.

3.8.1 INITIATE

The initiate flow diagram refers to the INITIATE major state. When the INITIATE flip-flop is set, INITIATE (1) H Λ STATE 0 H generates an output data break request at the address selected by the ADRS register. As with all enable states where a data break is requested, the timing generator is advanced at the trailing edge of address accept. When the timing generator is advanced, the four mini states are generated. The first mini state loads the data read from core (the first location of the APT table) into the most significant 12 bits of the A register (AMSW) and into the field bits of the appropriate address registers. The second mini state increments the ADRS register so that it points to the second location of the APT table. Mini States 3 and 4 are not used in this case. At the end of Mini State 4, State 0 is reset; at the trailing edge of the next clock cycle, State 1 is generated. (State 1 H Λ) (INITIATE (1) H) again requests an output data break at the address selected by the ADRS register. The trailing edge of address accept will again advance the timing generator and produce

the faur mini states. In this case, Mini State 1 is used to load the data from memory into the least significant 12 bits of the A register (ALSW). Note that the least significant 15 bits of the A register now cantain the initial setting of the FPP12 program counter (FPC). During (STATE 1Λ) (MINI STATE 2), the contents of the A register are transferred to the O register, and the ADRS register is incremented to point at the next location in the APT table. Mini State 3 is used to enable the transfer of the least significant 15 bits af the O register to the operand address register (OP ADDR) and the FPC is loaded from the OP ADDR during Mini State 4.

All the other registers are conditionally loaded from the APT depending on the state of the appropriate bit in the command register (refer to Table 1-3). In the case where the register is not loaded from the APT, the timing generator is advanced when the ENABLE state is entered and the only function performed during that time state is incrementing the ADRS register to the next location in the APT table.

At the end of the INITIATE cycle, the FETCH major state flip-flop is set, and the timing generator is set to State 0.

3.8.2 FETCH

A block diagram of the FPP12 FETCH flows is shown in Figure 3-7. (FETCH (1) HΛ) (STATE 0 H) generates an output data break request at the address specified by the FPC. Mini State 1 generates the enable that causes the data read from core to be loaded inta the FPP12 instruction register (FIR). During Mini State 1, a 27_8 is loaded in the A register and the exponent of the contents of the FPP12 accumulator (FAC) is loaded in the B register. The difference is then loaded in the O register during Mini State 2. The result that is stored in the O register is used only if a JAL instruction was fetched. In this case, the O register can be tested during Mini State 4 to see if the FAC exponent is greater than 27_8. This allows the JAL instruction to be treated the same as the other conditional jumps.

If FIR3 = 0 and FIR4 = 0, the FIR is decoded in a different fashion from that of the data reference instructions. Each of these special instructions has a unique flow diagram. The FIR and the FPP12 mode bit (CR0) determine which flow diagrams are pertinent.

a. LDA (FLDA)
b. STR (FSTA)
c. D.P. ADD & SUB (FADD, FADDM, or FSUB & Double-Precision Mode)
d. ADD & SUB of F.P. NOS. (FADD, FADDM, or FSUB & Floating Point Made)
e. MULTIPLY (FMUL or FMULM)
f. DIVIDE (FDIV)

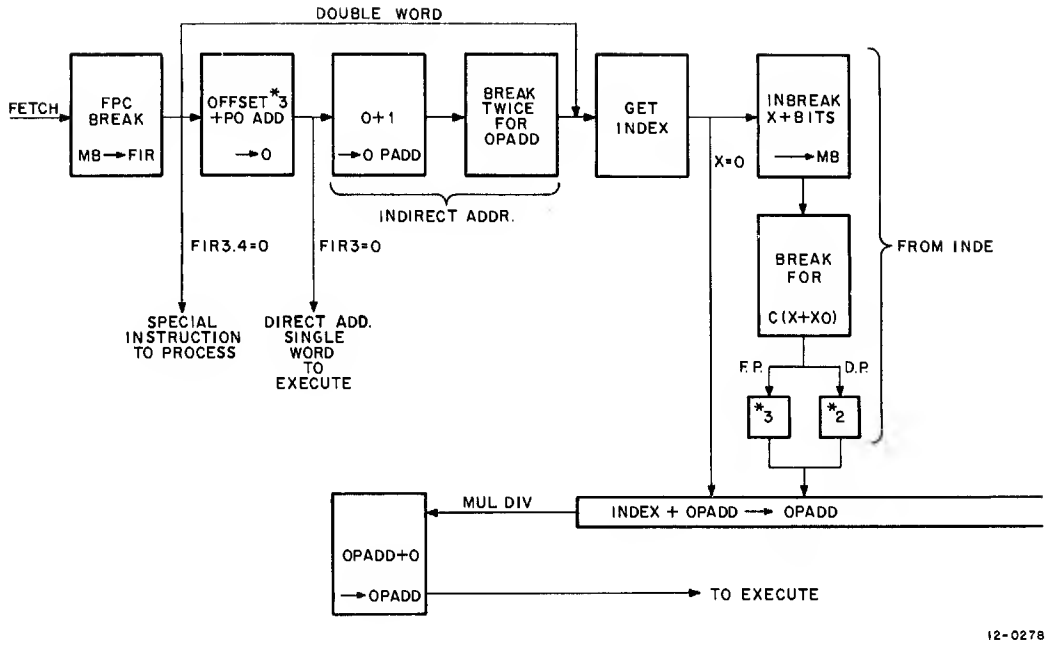


Figure 3-7 Block Diagram of FPP12 FETCH Flow

3.8.3 DEPOSIT

At the end of any arithmetic calculation, the FPP12 normalizes and rounds off the result. Along with the calculation of the exponent of the result, this is done during the DEPOSIT major state (see Figure 3-8). The DEPOSIT major state is enabled by causing the timing generator to jump to State 11.

When State 11 is enabled, the 0 register is shifted toward the most significant bit until the number is normalized; this is indicated in the flows as 0 (N) 0 (N-1). The 2's complement of the number of shifts required is tallied in the SHFT CNTR. When the number in the 0 register is normalized, the timing generator is advanced and the four mini states are generated. Normalization is performed by the clock that is free running even when the timing generator is not running. State 11 Mini State 1 clears the B register, loads the A register with the contents of the 0 register, and sets the CHECK FOR FLO flip-flop if the number in the 0 register is positive. State 11 Mini State 2 causes the rounding off of the result. The only case where overflow can occur is if the 0 register contained a 37777777 and the most significant bit of the 0 register extension was set. Hence, if the 0 register is negative after the rounding operation and the CHECK FOR FLO flip-flop is set, an overflow has occurred. If the FPP12 is in the double-precision mode, the overflow causes the FPP12 to exit. If the FPP12 is in floating-point mode and the rounding causes an overflow, the contents of the 0 register are shifted right one position and one is added to the exponent. This is done during State 12 Mini State 1 and 2. Since the 0 register cannot shift right, the contents of the 0 register are right-shifted through the multiplexer and loaded into the B register. The B register is then loaded into the 0 register and the

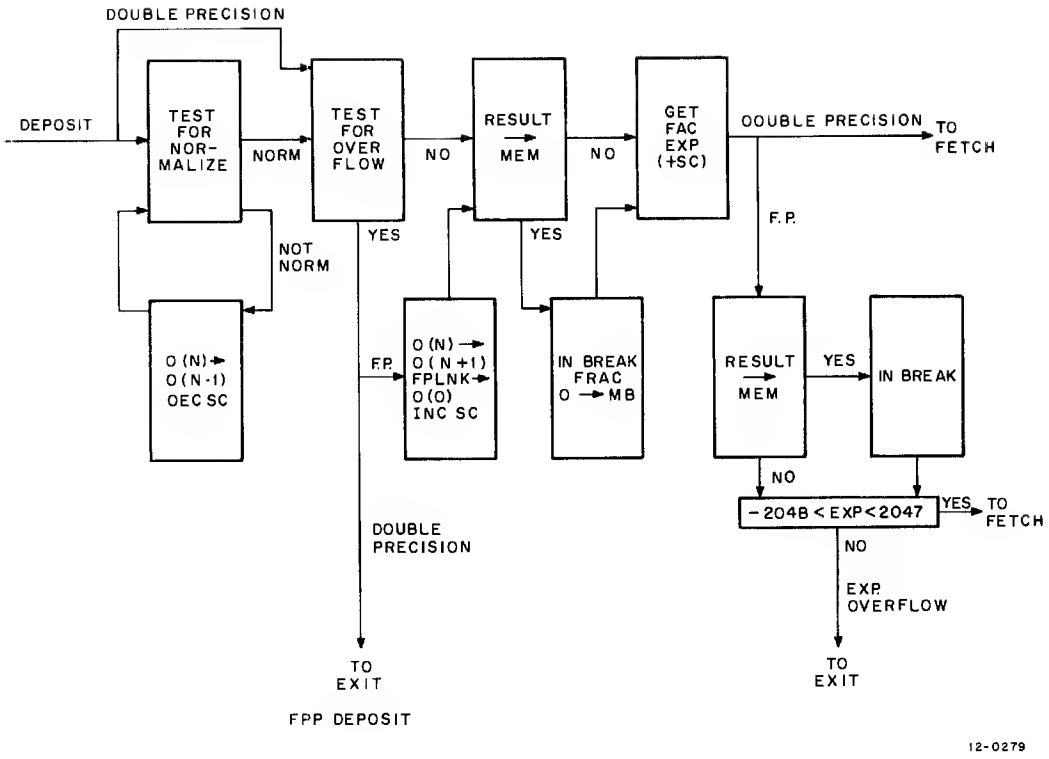


Figure 3-8 Block Diagram of FPP12 DEPOSIT Flow

SHFT CNTR incremented. The mantissa of the result is now in the 0 register; the number of left shifts that was required to normalize the 0 register is contained in the SHFT CNTR.

State 13 is used to store the least significant half of the mantissa in memory, in the case of an FADDM or FMULM instruction. During State 14, the most significant half of the 0 register is stored in memory in the case of an FADDM or FMULM instruction. The 0 register is loaded into the least significant 24 bits of the FAC for all other instructions during State 14. During State 14 of DEPOSIT, the value of the exponent of the normalized and rounded result is calculated. The exponent of the un-normalized result is assumed to be stored in the least significant 12 bits of the MQ (MQLSW) for all instructions except FMUL, FMULM, and FDIV. In the case of these three instructions, note that there are still two mini states (3 and 4) that can be used to generate enables so that the special instructions requiring only two mini states can be completed during FETCH and State 0. The other special instructions that require more time states cause FETCH to be reset and the PROCESS major state to be set at the end of State 0.

If FIR3 = 1 or FIR4 = 1, the instruction is decoded as a data reference instruction. In the case of a double-word instruction (FIR3 = 1 and FIR4 = 0), States 1, 2, and 3 are skipped, and State 4 is enabled after State 0. The operand address or the location of the indirect address for single-word instructions

is calculated during States 1 and 2, and the indirect address is fetched during States 3 and 4. The remaining time states in the FETCH cycle are used to modify the operand address with the contents of an index register if any index modification has been specified. In the case of the single-word indirect instructions, the indirect address is stored temporarily in the MQ (State 3 Mini State 2 and State 4 Mini State 1) and then loaded into the A register when it is required (State 7 Mini State 1).

There is also a flow diagram for each of the data reference instructions. When the OP ADDR has been set to the appropriate value, the FETCH flow diagram indicates a transfer to the EXECUTE major state. In this case, the EXECUTE flip-flop is enabled and the timing generator returned to State 0. The instruction in the operand exponent is assumed to be stored in the MQLSW. State 15 of DEPOSIT loads the exponent of the normalized and rounded result in the appropriate register and checks the value of the exponent to see if it is out of range; that is, less than 4000 or greater than 3777.

3.8.4 EXIT

The EXIT major state is discussed in detail in Paragraph 3.3. After INITIATE, the ADRS register is pointing at the highest memory location in the APT. During EXIT, the active registers are loaded into APT starting at the highest location and decreasing to the lowest.

The conditionol trapping of an exponent underflow error is performed in the EXIT major state; that is, all error conditions including exponent underflow cause the EXIT major state to be set instead of the FETCH major state. During EXIT State 0 the cause of the exit is checked to determine if it was an exponent underflow. If bit 1 of the command register is set, the usual exit routine is followed. If, on the other hand, bit 1 of the command register is not set, the FAC or the operand (for ADDM or MULM) is set to 0 and the FETCH major state is set; this allows the program to proceed.

3.8.5 LDA and STR

The LDA flow diagram is executed during the EXECUTE major state of an FLDA instruction. If the FPP is in floating-point mode, the contents of the three sequential memory locations defined by the contents of the OP ADRS register are loaded into the FAC. In double-precision mode, only the contents of two sequential memory locations are loaded in the FAC. The STR flow diagram is implemented during the EXECUTE major state of a FSTA instruction. In this case, the appropriate 12-bit bytes of the FAC are stored in the memory locations defined by the OP ADRS register.

3.8.6 DP ADD and SUB

This flow diagram refers to the EXECUTE state of an FADD, FADDM, or FSUB instruction, when the FPP12 is in double-precision mode. If the result is greater than $37777777)_8$ or less than $40000001)_8$

the fraction overflow bit (bit 4 of the status register) is set and the EXIT major state is enabled causing the FPP12 to halt.

3.8.7 ADD/SUB of FP NOS

This flow diagram is implemented during the EXECUTE major state of an FADD, FADDM, or FSUB instruction, when the FPP12 is in floating-point mode. In order to add or subtract two floating-point numbers, the exponents must be aligned; that is, the fractional part of the number with the smallest exponent must be shifted right and the exponent incremented until the two exponents are equal.

Since the A register cannot be shifted, the fraction of the number with the smallest exponent must be loaded into the B register. This is done in the following fashion. During State 0 Mini State 1 and 2 the difference between the FAC exponent and the operand exponent is calculated and stored in 0. The operand exponent is stored in the MQLSW for future reference. The sign of the difference is used to determine which exponent is larger and, hence, which fraction must be loaded into the B register. The absolute value of the difference is also loaded into the SHFT CNTR which is subsequently used to determine the number of times the B register is shifted. During State 0 Mini State 3 and 4, the number of shifts required is checked to determine if it is more than $27)_8$; that is, if the fraction to be shifted will be completely shifted out of the B register. The OVERSHFT flip-flop, which is set during State 1 Mini State 2, is used to indicate that the required number of shifts is greater than $27)_8$.

State 1 and State 2 are used to fetch the most significant word of the operand fraction. In the case of FSUB, where the operand exponent is greater than the FAC exponent, the 1's complement of the operand fraction is loaded into the A. This is necessary since the ALU cannot perform the operation B minus A.

Before the B is shifted, the fraction of the number with the largest exponent (which is stored in the A register) is checked to make sure that it is nonzero. This prevents the loss of significance when a number with a nonzero exponent and a zero fraction is added to or subtracted from another number. If the fraction of the number with the largest exponent is zero, the fraction stored in the B register is loaded into the 0 register and its associated exponent is loaded into MQLSW and the DEPOSIT major state is enabled.

When State 3 is enabled, the B register is shifted toward the least significant bit. The number of positions is determined by the number contained in the SHFT CNTR. When shifting is completed, the timing is advanced and Mini State 1 is used to perform the required operation between the A and B registers. If the operation causes an overflow condition, adjust the fraction one bit position right and set the SHFT CNTR to 1.

At the end of State 3, the result of the addition or subtraction of the aligned fractions is stored in the 0 register, MQLSW contains the value of exponent of the aligned fractions, and SHFT CNTR contains a 0 or contains a 1 in the case when the fraction of the result was shifted right. The DEPOSIT major state is entered to perform the normalization, rounding, exponent calculation, and storage of the result.

3.8.8 MULTIPLY

The MULTIPLY flow diagram details the algorithm used by the FPP12 to perform floating-point and double-precision multiplications. The absolute values of the two fractions are multiplied together to give a positive result which is negated if either but not both of the fractions are negative. The absolute value of the operand fraction is loaded in the MQ during State 0 and State 1. If the FAC fraction is negative, the complement is loaded in A and a CARRY INSERT is generated when A and B are added during the multiply cycle so that the 2's complement of the FAC fraction is added to the contents of the B register.

The multiplication of the two fractions is performed in State 2. Each cycle of the algorithm requires two clock pulses. The first clock pulse is used to load A plus B into the O register and to decrement the SHFT CNTR. The second clock pulse loads the partial sum divided by 2 into the B register if the 23rd bit of the MQ is 1. If the 23rd bit of the MQ is 0, the previous partial sum that is contained in B is divided by two. The second clock pulse also shifts the MQ toward the least significant bit, which brings the next binary bit of the multiplier into the 23rd position for testing on the next cycle. The final product is stored in the B register.

The remaining time states used in MULTIPLY store the product in O and load the operand exponent in the MQLSW for use in the DEPOSIT cycle.

3.8.9 DIVIDE

The Divide algorithm used in the FPP12 is shown in the DIVIDE flow diagram. Again, the divide routine makes both the divisor and the dividend positive, calculates the quotient, and negates it if either but not both of the divisor or the dividend was negative.

During State 0 and State 1, the FAC and the absolute value of the operand fraction are loaded into A and the B registers, respectively. During State 2, the division of the fractions occurs. Again, like the MULTIPLY algorithm, two clock cycles are required for a shift cycle of the divide algorithm. During the first half of the cycle, the divisor is subtracted from the current remainder (which is stored in the A register) and, if the result is positive (CARRY OUT = 1), the difference is loaded into the

O register and a 1 is shifted into the least significant bit of the MQ. During the second half of the cycle, the new remainder is multiplied by two and stored in the A register. The O register is also shifted left so that the cantents of the A and the O registers are the same. If the result of the trial subtraction is negative (CARRY OUT = 0), a zero is shifted into the least significant bit of the MQ and the current remainder, which is stored in both the O and the A registers, is multiplied by two. This multiplication is done by shifting the O register. The second half of the cycle then loads the cantents of O register into the A register. The 28-bit quotient is found in the MQ when division is complete. The remaining time states used in the divide routines load the MQ into the O, and the operand exponent into the MQLSW. During State 2 Mini State 3, the quotient is divided by two and a one is loaded into the shift counter if the first subtraction gave a positive result. This will occur if the dividend is greater than the divisor.

3.8.10 SPECIAL INSTRUCTIONS

The remaining flow diagrams are those associated with the instructions that use Special Format 1, 2, and 3. Four of these instructions (FEXIT, FCLA, STARTF, and STARTD) are performed completely in FETCH State 0. All of the conditional jumps are also completed in FETCH State 0, if the condition is not true; that is, if the jump is not performed. In all other cases, the PROCESS major state is entered at the beginning of State 1. Table 3-2 shows the equivalence between the instruction mnemonic and its corresponding flow diagram heading.

3.9 MAINTENANCE LOGIC

Maintenance logic, built into the FPP12, permits the CPU to examine, in detail, the operation of the FPP12. For instance, the CPU can issue IOTs that force the FPP12 to cease operation after every major time state. Other IOTs permit the CPU to examine internal registers in the FPP12. Using these toals, a diagnostic program can pinpoint the exact step in the flow charts in which the FPP12 fails. This shauld isolate the failure to within one or two gates. Diagnostic instructions can sometimes be used to debug programs. For instance, there is a maintenance instruction that reads the 12 least significant bits of the APT pointer. If a program has more than one APT it can be desirable to determine which APT is currently in use. This can be done by issuing the maintenance IOT 6565 with the AC clear.

A complete list of maintenance IOTs and their functions follows. Data from the FPP12 is inclusively ORed into the AC when the maintenance mode IOTs are used.

Octal Code	Mnemonic	Function
6561	Enter Maintenance Mode or Maintenance Step	a. This IOT is typically issued prior to FPST to begin maintenance mode. b. 6561 is issued when halted at the end of a major time state to cause the advance to the next time state. c. Maintenance mode is cleared whenever the FPP Interrupt Request flag is cleared.
6562	Read States	The current major time state and enable state are ORed into the AC, according to Table 3-3.
6563	Read OMSW	OR the OMSW register into the AC.
6564	Read OLSW	OR the OLSW into the AC.
6565	Read APT	OR the least significant 12 bits of the APT pointer into the AC.
6566	Read MQLSW	OR MQLSW into the AC.
6567	Load Shift Counter	Load the shift counter with the significant 6 bits of the AC.

Maintenance Instructions are detailed on print D-BS-FPP12-0-C12.

Table 3-2
Equivalence Between Instructions and Flow Routines

Instruction	Flow Diagrams
FEXIT	EXT
FPAUSE	PSK
FCLA	CLR
FNEG	NEG
FNDRM	NRM
STARTF	STF
STARTD	STD
JAC	RTN
ALN	ALN
ATX	ATX
XTA	XTA
FNOP	NOP's
LDX	LDX
ADDX	ADX
JEQ	JMPS

(Continued on next page)

Table 3-2 (Cont)
Equivolence Between Instructions and Flow Routines

Instruction	Flow Diogrom
JGE	JMPS
JLE	JMPS
JA	JMPS
JNE	JMPS
JLT	JMPS
JGT	JMPS
JAL	JMPS
SETX	MUX
SETB	MVP
JSA	JSB
JSR	JMK
JXN	JXN

Table 3-3
Definition of AC Bits After IOT 6562 Read States

AC Bit	Function
00	Most significont bit of major time state counter
01	Bit 1 of major time stote counter
02	Bit 2 of major time state counter
03	Bit 3 of major time stote counter
04	CRN deposit flop (1) H
05	CNR fetch flop (1) H
06	CRN execute flop (1) H
07	CNR exit flop (1) H
08	CNR initiate flop (1) H
09	CNR process flop (1) H
10	AST0 shft FAC FRAC H
11	AST1 no shft (1) H

CHAPTER 4
FPP12 INSTALLATION

4.1 INSTALLATION

The Floating Point Processor is a standard PDP-8 type, data break I/O bus peripheral. The FPP12 is attached to positive bus computers with BC08B cables and to negative bus computers with BC08D cables, according to drawing D-UA-FPP12-0-0. Standard FPP12 logics are wired for PDP-12 computers. Slight wiring alterations for PDP-8/I, PDP-8/L, PDP-8/E, PDP-8, and LINC-8 Computers are normally made as the units are checked out in the factory. The wiring changes for field conversion are shown in Tables 4-1 and 4-2. It is also necessary to exchange six modules when converting the FPP12 from a positive to negative I/O bus. These modules and their locations are shown in Table 4-3.

4.2 AC POWER REQUIREMENTS

Typically, the FPP12 is shipped in a standard DEC 19-in. cabinet. The AC power for the H721 Power Supply is controlled through a H854 (H854B for 50 Hz) Power Control. This power control is connected to the AC power with a line cord terminated in a Hubbell 30-amp twist-lock male plug.

Total power consumption for the FPP12 logic and power supply is 150W.

Table 4-1
PDP-8/L, PDP-8/I Positive Bus and PDP-8/E

Name	Run	Add	Delete
CI1 ADD ACC (1) L	D04D1 - D30N1		X
EXT ENAB INT PAUSE H	F05U2 - B03V2		X
CI1 BIS 05 (1) H	D30N1 - A11T2	X	

Table 4-2
PDP-8, LINC-8, and PDP-8/I with Negative Bus

Name	Run	Add	Delete
EXT ENAB INT PAUSE H	F05U2 - B03V2		X
CI1 IOP 1 H	C01M2 - E01M2		X
	A01H1 - C01M2		X
	C01M2 - E01M2	X	
	C01M2 - A10F1	X	
CI1 IOP	C01N2 - E01N2		X
	C01N2 - A10P2		X
	C01N2 - E01N2	X	
	C01N2 - A10N1	X	
CI1 IOP H	C01P2 - E01P2		X
	C01P2 - A10S1		X
	C01P2 - E01P2	X	
	C01P2 - A10R1	X	
CI1 INIT L	B31P2 - A21D1		X
CI1 INIT H	B31P2 - A21E1	X	
AI1 ADD ACC (1) L	A11V2 - D04D1		X
	D04D1 - D30N1		X
	D04D1 - A11V2	X	
CII BTS 05 (1) L	D30N1 - A11S2	X	

Table 4-3
Module Changes for Negative Bus Computers

Slot	Positive Bus Modules	Negative Bus Modules
B08	M101	M100
C3	M623	M633
C4	M623	M633
C5	M623	M633
D05	M623	M633
E3	M101	M100
E4	M101	M100
E8	M623	M633
F05	M623	M633

4.3 CHECK OUT

Diagnostic programs necessary for checking out the FPP12 are as follows:

FPP12 Instruction Test 1	MAINDEC-12-D0LA
FPP12 Instruction Test 2A	MAINDEC-12-D0MA
FPP12 Instruction Test 2B	MAINDEC-12-D0NA
FPP12 Instruction Test 2C	MAINDEC-12-D0OA
FPP12 Address Test	MAINDEC-12-D0PA
FPP12 Exerciser	MAINDEC-12-D0QA

These diagnostic programs should be run in accordance with instructions packed with the diagnostic programs. On the PDP-12 Computer, it is advisable to run the system program for Display Oriented Research Analysis (DORA). DORA is part of the Analytical Instrumentation Package Operating System (AIPOS). AIPOS and DORA will run on any PDP-12 Computer equipped with 8K of core memory, the AD12 analog knobs, and the FPP12. Instructions for the operation of DORA are contained in a separate manual (DEC-12-SQ1A-D).